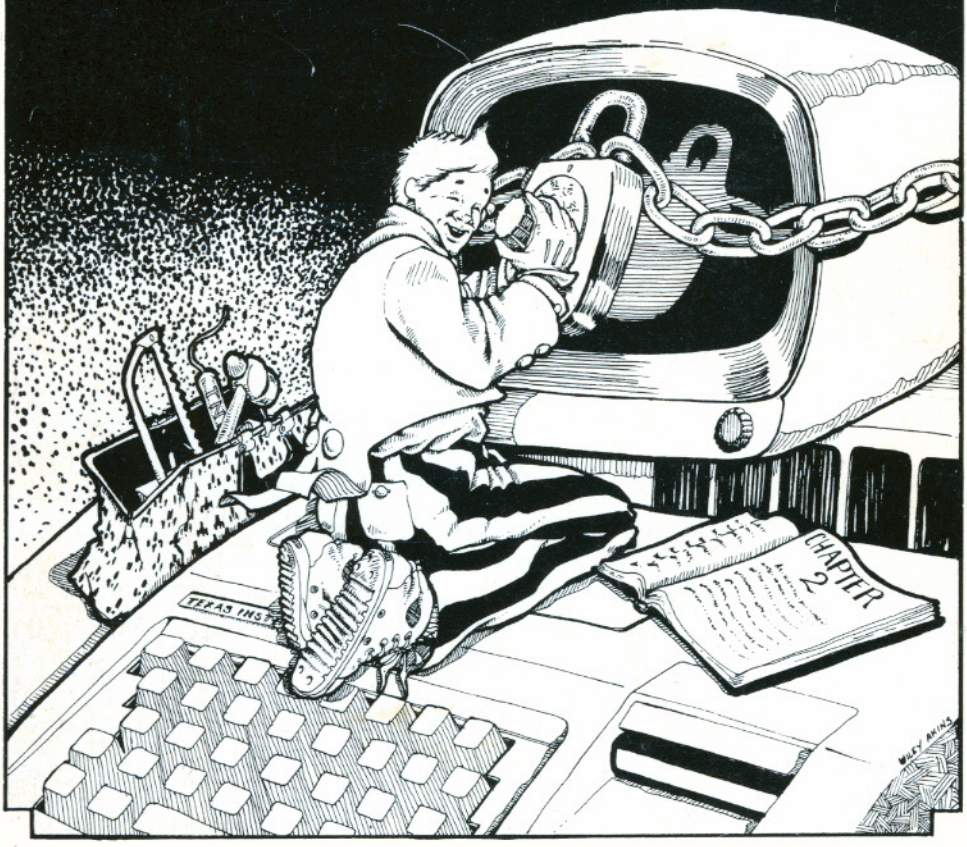
CRACKING THE 99/4A



Serious fun for the home computer enthusiast...

by Brian Prothro

CRACKING THE 99/4A

BRIAN PROTHRO

Published by Midnight Express
Box 26941
Austin, Texas 78755

ISBN 0-917915-00-3

To
"The Boy"

Artwork by
Wiley Akins

AND

Thank you Linda Lewis,
what's in a name?

FORWORD

Computers seem to get a lot of bad press these days. I'll admit that they often come equiped with a maze of cable assemblies and the operator manuals read like War and Peace in Swahili. But aside from all the publicity, computers are not so bad. They can do amazing feats of magic like play munchman, multiplan, or beat I.R.S. and aside from eating a disk or tape now and then, they are relativly easy to live with. Computers make great companions!

Whether you are just learning BASIC, or are well on your way to understanding programming, this book offers you the opportunity to get to know your 99/4A. Often the best solutions to your programming needs are learned by studying how others have solved their own programming problems. For the most part, this book is not a BASIC tutorial, but offers an opportunity to to learn while exploring finished programs, as well as add to your software library.

The various submissions included in this book stretch from games to very handy programing utilities in assembly language. You will find that the majority of these programs and tips require extended basic and some, additional hardware such as the 32K expansion memory. The hardware requirements for each program are printed at the beginning of each program description.

CONTENTS

TUTORIALS

GAMES

HOME

SPEECH

UTILITY

PROGRAMMING TUTORIAL

BASIC PROGRAMMING

PART I

The tutorial will cover what I call a data manipulation program. This type of program will allow its user to enter information, change it if desired, save the data to a file, and reload it for later use. This is a type of program that businesses frequently use. But don't let that stop you from reading on! The techniques described here can be used by anybody who has data to store, whether it's data from a scientific experiment or a list of monthly expenses.

Some things covered here are; files with cassette, structured programming, and linked lists. To keep this discussion general, the program will be written in standard console (i.e., not Extended) BASIC and use a cassette recorder to store the data. Part 1 will deal with defining the data the program will manipulate, how it is stored on the file, and how to get it back.

The first thing that a programmer should do before beginning to write a program is to decide what the program should accomplish and how it is to be done. This is very important, because you should know where a journey will end before it is begun. How else will you know when you've finished? Describing what a program should do is also useful because other people who might use it can examine the design. Suggestions can be made while changes are easy to make, before the program is written.

The program to be described in this tutorial is a mailing address program. When it is finished, it will be able to do the following:

1) allow the user to keep an alphabetical list of addresses
2) be able to find an address using only a name
3) scan the addresses in the list
4) make changes in any listed address
5) keep all information on a cassette file
6) keep the following information for each address: name, two lines of street address, city, state, zip code, and phone number

First we'll deal with how the data should be stored in memory and on file.

Now that we know what data will be stored for each address in the list, we need to decide how to store that information in memory. The best method is to use a two-dimensional array. Imagine a two-dimensional array as a large blackboard that has rows and columns marked on it. In our case, each row on the blackboard will be used by one of the addresses to store the information. Each column on the row, then, is the location where a part of each address will be stored. For example, the first column of the row will contain the name, the second column will contain the first line of the address, the third column the second line of the address, the fourth column the city, the fifth column the state, the sixth column the zip code, and the last column will contain the phone number.

Within the BASIC program, it is easy to access a particuliar piece of information using a two-dimensional array. For example, if we want to print the phone number of the fifth address on the list, we can simply use the code:

```
PRINT ADDR$(5,7)
```

A two-dimensional array has a name like any other variable. It is followed by two numbers (thus the name two-dimensional) in parenthesis. The first number is the row to access, the second in the column within that row. From there on, the information located at the requested row and column is handled like any other variable in BASIC.

In our program, all address information will be stored in the array named ADDR$. In this first draft of the program, it will perform the following tasks:

1) allow us to enter information into the mailing address array ADDR$
2) save that information into a file
3) read that information back into memory from the file
4) redisplay the information

The program will become more complex as we go along.

This is a listing of the controlling part of the program:

```
100 REM A SIMPLE MAILING ADDRESS PROGRAM
 BY MIKE SCHULTZ
110 REM WRITTEN 7/1/1983
120 REM CRACKING THE 99/4A
130 OPTION BASE 1
140 DIM ADDR$(20,7)
150 CALL CLEAR
160 LSTADR=0
170 GOSUB 220
180 GOSUB 390
```

```
190 GOSUB 500
200 GOSUB 610
210 END
```

Notice the four gosubs. They correspond to the four tasks that I described above. Notice also the DIM statement. It tells BASIC that the array ADDR$ is going to have 20 lines and 7 columns. We know that each line is going to be used for a separate address. Finally, notice the variable LSTADR. At any point in the program this variable will contain the last row of ADDR$ that stores useful information. The program will change LSTADR whenever a new address is added to the list. This is the subroutine that will put information in the array. Note that this isn't going to be the final version of this subroutine.

```
220 REM THIS ROUTINE ACCEPTS NAMES AND A
DDRS
230 INPUT "ANOTHER ADDRESS? ":Q$
240 IF Q$="Y" THEN 260
250 RETURN
260 LSTADR=LSTADR+1
270 GOSUB 290
280 GOTO 230
```

This routine will continue to execute till the user answers the question "ANOTHER ADDRESS?" by anything other than Y. This is not necessarily the best way to exit such a loop, but since it will be rewritten later, it will do for now. (It would be better if the program continued when Y is entered, returned to the controlling routine when an N is entered, and reasked the question when anything else is entered.)

Notice that LSTADR is increased by one when another row is added to the list. The routine located at line 290 asks the user for the information to be stored on the row:

```
290 REM THIS ROUTINE ACCEPTS A SINGLE NA
ME
300 REM  AND PLACES IT IN THE ROW INDICA
TED BY "LSTADR"
310 INPUT "NAME: ":ADDR$(LSTADR,1)
320 INPUT "ADDR: ":ADDR$(LSTADR,2)
330 INPUT "ADDR(2): ":ADDR$(LSTADR,3)
340 INPUT "CITY: ":ADDR$(LSTADR,4)
```

```
350 INPUT "STATE: ":ADDR$(LSTADR,5)
360 INPUT "ZIP: ":ADDR$(LSTADR,6)
370 INPUT "PHONE: ":ADDR$(LSTADR,7)
380 RETURN
```

Notice how LSTADR is used. This routine assumes that the calling routine has correctly assigned LSTADR to the row where the information will be stored. The INPUT statements simply ask the correct questions and make sure that the data gets stored in the correct columns. When all the data is entered, answer N to the question ANOTHER ADDRESS. The controlling routine then calls the routine that writes the data to the cassette file.

Before we examine the routine that writes to the file, a few words about files in general are necessary. The concept of a file is very similiar to the files in a filing cabinet. Inside a file folder are pieces of paper containing information that we wish to save (such as addresses). We want to save the information because it may prove useful or necessary.

In a computer file, the equivalent of the pieces of paper in the file are called records. The information placed in the record is converted to electrical impulses instead of ink marks on paper. The computer is capable of creating these records on the cassette tape the same way that we are capable of recording information on paper. The computer is also capable of reading this information back into its memory.

Records have a length — the number of characters (bytes) that can be stored in each. In general, records have a either fixed or variable size (just like paper), but cassettes are very simple devices and are limited to a fixed length record. Disk drives are more complicated and thus can have variable length records. This is not a major problem with cassettes, however, as the difference between a fixed or variable length record is basically unimportant.

In a paper file, people can randomly flip through the file looking for a particular piece of paper. There are random files that the computer can use, but these files are available only on disk drives. The other way for the computer to examine the records is sequentially; that is, to start with the first record on the file, do something (or nothing) with the data, and then proceed to the next record on the file. For the purpose of our program (since we want to read every record into memory), sequential files will be fine.

Let's look at the routine to write our address array to the cassette file.

```
390 REM THIS ROUTINE SAVES THE ADDRESS T
O FILE
```

```
400 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,OU
TPUT,FIXED 192
410 PRINT #1:LSTADR
420 FOR I=1 TO LSTADR
430 FOR J=1 TO 6
440 PRINT #1:ADDR$(I,J),
450 NEXT J
460 PRINT #1:ADDR$(I,7)
470 NEXT I
480 CLOSE #1
490 RETURN
```

This routine deserves careful examination. First, notice the OPEN statement. The cassette file is oPened as sequential because that is the only type of file that a cassette can handle. It it opened as INTERNAL because this file will be read by the program again (but at a later time). The INTERNAL designation makes reading the information back into memory a lot easier.

The file is opened as OUTPUT so that the user of the program will be asked to press the RECORD switch on the recorder.

The file is opened as FIXED because the cassette can handle only FIXED length records. It is set up with 192-byte records because that is the maximum record for the cassette and will handle all situations. Line 410 writes the value of LSTADR to the file, on a record all by itself. This will let the program know when it is run later how many records of actual information ADDR$ contained on this run.

The rest of the routine writes the information of the array ADDR$ to the cassette file, one record containing one array row. On line 440, the comma at the end of the line is a signal from the program to BASIC that it is not through writing the record. It is the PRINT statement on line 460 that actually signals the end of the record. After this line, BASIC will advance the cassette tape to the next record.

Notice that the program is not making any attempt to separate the information in the record by inserting spaces. Opening the file as INTERNAL made this unnecessary. BASIC will put the needed information in the file so as to separate the columns during input. After writing all the records, the CLOSE command allows us to tell the user to turn off the cassette recorder.

How is the information read back into memory later? It is simply the reverse of the output process.

```
500 REM THIS ROUTINE READS THE ADDRESSES
 FROM FILE
510 OPEN #1:"CS1",SEQUENTIAL,INTERNAL,IN
PUT ,FIXED 192
520 INPUT #1:LSTADR
530 FOR I=1 TO LSTADR
540 FOR J=1 TO 6
550 INPUT #1:ADDR$(I,J),
560 NEXT J
570 INPUT #1:ADDR$(I,7)
580 NEXT I
590 CLOSE #1
600 RETURN
```

Notice that the OPEN statement describes the file in essentially the same way as in the output routine, except that INPUT is used.

The value of LSTADR is read from the file first so that the program can know just how many lines of address information was stored on the file.

Next, the information is read from each record back into the array ADDR$.

And finally, here is a simple routine to display the information that has been read in from the file.

```
610 REM THIS ROUTINE PRINTS THE INFORMAT
ION IN ADDR$
620 FOR I=1 TO LSTADR
630 PRINT "NAME: ";ADDR$(I,1)
640 PRINT "ADDR: ";ADDR$(I,2)
650 IF ADDR$(I,3)="" THEN 670
660 PRINT "      ";ADDR$(I,3)
670 PRINT "CITY: ";ADDR$(I,4)
680 PRINT "STATE: ";ADDR$(I,5);" ZIP: ";
ADDR$(I,6)
690 PRINT "PHONE: ";ADDR$(I,7)
700 PRINT
710 NEXT I
720 RETURN
```

This is not a complete program. You'll notice that the program is not very flexible. There is no way to change anybody's address. The addresses

are not in alphabetical order. Finally, the routine that prints the addresses to the screen prints all of the addresses instead of allowing the user to be selective. These (and other problems) will be fixed later in this tutorial.

PART II / STRUCTURED PROGRAMMING

Now we'll discuss more about the overall planning of a program and how to make a more flexible control routine.

You may recall that there are four tasks the mailing address will do: 1) allow us to enter the addresses to be kept by the program; 2) save that information to a file; 3) read the saved information from the file back into memory; 4) search the addresses stored in memory for a particular name. The last section presented simple versions of solutions to all these tasks in the form of BASIC subroutines. This concept of breaking a job into tasks is worth a closer look as it is a major cornerstone for modern programming.

This technique, called modular programming, actually has roots in our everyday life. When we are presented with something to be done, it is rarely a single simple problem, although we usually don't realize it. Take going to the store to buy milk for example. This is in itself not a single task, but at least two: 1) going to the store and 2) buying milk. Even the going to the store task is not a simple one. It itself is made up of many subproblems involved with locating the car, getting it started, and so on, not to mention the many problems encountered while travelling to the location, and finally locating a parking place. And yet, people solve these types of problems regularly without conscious thought.

The same techniques can be applied to programming. When we have a problem whose solution we would like to automate on a computer, we must first know how we would solve that problem ourselves. If we don't know that, then we must begin by breaking the problem down into smaller pieces. Every time a problem is broken into pieces we must ask ourselves if we know how to solve this piece of the problem in BASIC. If the answer is yes, then this piece is small enough. If the answer is no, then we must try and split the problem some more.

While we are performing this divide and conquer process, it is helpful to know three things for each piece of the puzzle. 1) What will have occurred before we get to this piece. 2) What will this piece do to help solve the problem. 3) What will the state of things be after this piece has been performed.

These things are very important to know because they can point out problems in the solution. For example, looking at the way one piece of the solution left things and how the next piece expects things may show us that the first piece isn't doing the whole job for us.

Now at this point you may be scratching your head, asking what in the world I'm talking about, so let's relate it back to the store example. The piece we'll take for the example is the actual buying of the milk. We know several things about what we've done at this point. We know that we have successfully completed traveling to the store. We also know that we have located the milk, picked it up, and gone to the checkout line. In this piece of the problem we then pay for the milk, and leave the store. We are now in the car ready to go home.

This may seem to be a large amount of unnecessary work to some people, but my experience has been that learning programmers get the most confused and frustrated when they are trying to write code and they have no idea where in the solution they are, or what has happened in the program before this point.

Now, how does this relate to BASIC? As I said earlier, each piece of the problem will eventually become a BASIC subroutine. The piece is entered via a GOSUB statement, does its job, and returns to the calling routine (piece) via a RETURN command. Since a particular subroutine may be itself made up of individual pieces, it calls the subroutines for theses pieces for them to do their job.

So, subroutines can call other subroutines, which in turn can do the same. This is all well and good, but somewhere there has to be a limit to all this. There has to be a routine that calls the major subroutines. This is the routine that is entered first when the user of the program types RUN and should be the routine that returns control of the computer back to the user. This is called the main program or controlling routine.

There are many techniques used for making controlling routines. The technique used depends on the problem to be solved. A purely mathematical problem would require the program to perhaps ask for some input from the user and then begin calculating. This is essentially the type of controlling routine that our mail address program currently has. As you can see, it is not well suited for this type of program.

For the data manipulation type of problem we are considering, the subroutines act as services that the whole program can perform for the user. The controlling routine's job, then, is to ask the user how it may be of service and call into action the appropiate subroutine to do the job. Thus, this type of program does not solve one specific problem (such as find Joe Cool's phone number for me) but many problems. In essence, the user of the program is still in charge of solving the problem and the program is providing tools for the solution.

The kind of controlling routine that I am going to show here is called a menu-driven program. That is, it presents the user with a list of options that they may chose at this time, takes the request from the user, and calls the subroutine to do the job. When the subroutine has finished, it returns to the controlling routine, which then asks the user what they would like next. Eventually, the user will be finished with the program and tell it to quit.

Our mail address programs will do the following things:

1) Since the program must have the addresses loaded into memory before it is any good, it will call the subroutine that reads the file first. Just in case the user wants to create a new address file, it will ask about this before calling the read file subroutine.

2) Next, it will ask the user if they want to modify addresses in the list, search the list for a name, or quit. When the user has made a valid choice, the approiate routine is called. After the subroutine returns, the controlling routine asks what's next? When the user finally says to quit, the controlling routine calls the subroutine to write the file (in case any changes were made to the addresses) and then terminate.

Now, our subroutines from part I won't do all of these things, but that's okay. They do enough for us to demonstrate that the controlling routine works properly. Later we'll complete the subroutines.

Here is the controlling routine. Notice the line numbers. The routine can be typed over the top of the existing control routine to replace it.

```
100 REM A SIMPLE MAILING ADDRESS PROGRAM
 BY MIKE SCHULTZ
110 REM WRITTEN 7/30/1983
120 REM CRACKING THE 99/4A
130 OPTION BASE 1
140 DIM ADDR$(20,7)
150 CALL CLEAR
155 PRINT "  THE MAILING LIST MANAGER":
: : : : : : : : : :
156 INPUT "READ IN LIST FROM TAPE? ":Q$
157 IF Q$="Y" THEN 165
158 IF Q$<>"N" THEN 156
160 LSTADR=0
162 GOTO 170
165 GOSUB 500
```

The code above displays a friendly message to let the user know that the program is running and then asks about reading in the addresses from a data file. It takes action based on the user's response.

```
170 CALL CLEAR
171 PRINT " THE MAILING LIST MANAGER"
172 PRINT : :" M - MODIFY THE LIST": :"
S - SEARCH THE LIST": :" Q - QUIT THE MA
NAGER": : : :"SELECT A FUNCTION:"
173 CALL KEY(0,F,S)
174 IF S=0 THEN 173
175 CALL HCHAR(23,22,F)
176 C=POS("QMS",CHR$(F),1)
177 IF C>0 THEN 180
178 CALL SOUND(500,220,0)
179 GOTO 173
```

This code displays the functions that the user can select and then waits for the user to press the letter of the function desired (lines 173 and 174). That letter is then redisplayed for the user (in line 175). The code at line 176 is a very nice way in BASIC of deciding if the letter selected is a valid one. Notice that "QMS" are all letters of the valid functions. The POS command will return the number that describes the selected letter's position in the "QMS" list; this number will be placed in the variable C. Thus it tells us if the letter pressed was the first letter in "QMS" (Q), or the second (M), or the third (S). If the letter selected is none of these, then POS will have the value 0 and the controlling program can sound an error tone and ask for a correct command.

```
180 IF C>1 THEN 190
185 GOSUB 390
186 END
```

This code is entered when the first letter in the list of valid commands is entered (Q for Quit). The subroutine to write the file is called. Then the program terminates.

```
190 ON C-1 GOSUB 220,610
191 GOTO 170
```

And, finally, this code calls the appropriate subroutine for the other functions. Notice that the variable C still has the position of the selected function's letter from the "QMS" list in the POS command. The ON GOSUB

command uses the value following ON to pick the correct line number from the list following GOSUB. A normal GOSUB is then performed to that line number. In our program, this command goes to the correct subroutine for the function selected.

When the routine finishes (executes a RETURN), the next line of code executed is 191. This line takes us back to the code that displays the menu.

```
715 INPUT "LIST COMPLETE: ":Q$
```

This line of code makes the search routine wait after it has displayed the list before returning to the controlling routine, which will then quickly clear the screen. This will give the user a chance to see the display before it disappears.

The subroutine doesn't have to do anything but print a message that it has been entered and return to the menu. Also notice that the order in which the selection letters are displayed on the menu doesn't have to correspond with the order in which they are listed in the POS command.

PART III / LINKED LISTS

Now I will discuss how to sort (alphabetize) the names. There are a number of ways of doing this and I considered many before settling on one as the best for this series. It is a little more complicated to explain than most, but will make a better program in the long run.

It also means that some major changes will have to be made in the rest of the program as it was developed in previous sections, so the entire program is reproduced here again.

Most of the methods for sorting data involve the program first accepting all the data, then performing the sort. On small computers such as the /4A, this can mean a several minute to several hour wait. The method described here, however, doesn't really sort the names so much as it keeps the names in an alphabetical list as they are entered. This will mean a slight delay between entering names into the list, but not enough to be a problem.

This method is called a LINKED LIST. It works like this: along with the information that is kept for each entry, the location within the list of the next person (in alphabetical order) is also kept.

In the previous version of the program, every row of the array ADDR$ could be thought of as a line on a blackboard or piece of paper (although it really in memory) containing a person's name. Now, when another name

needs to be inserted into the list of names, if the list is already in alphabetical order, we simply search down the list until we find the proper place, and insert the new entry. The only problem with this is that now we must move all lines below where the new entry is inserted down a line on the imaginary blackboard or paper. Moving data around in the computer is like copying the lines by hand for us; it's a very slow process. (In fact it is the very reason that most of the other sorting methods are so slow.) So how are linked lists better?

They're better because we don't have to move data around. Instead each line, along with the person's name and address, contains the line number of the "next person" (alphabetically) on the list. When we add a person to the list, we put their information on the next available line on the blackboard. We then find the name of the person who will come just before the new entry in the list, and change their "next person" column to contain the new entry's line number. Similiarly, we find the name of the person that will follow the new entry in the list, and place the line number of the next person into the new entry's "next person" column.

Let us see an example of this. Below is a short list of names. The left-most column is the line number of the entry, the middle column is the person's name, and the last column is the line number of the name following this entry in alphabetical order.

Line Number	Name	Next
1	Jill	5
2	Sam	Ø
3		
4	Ann	1
5	John	2

In this example, Ann on line 4 is the first person in the list. (We'll talk later about how we knew this.) If we look at the "next" column, we find that Jill in line 1 is next, then John on line 5, and finally Sam on line 2.

We knew that Sam was last because when we were building the list, we made sure that the last person on the list had a Ø in their next column.

When we write a program that uses a linked list, we also have to keep a variable that contains the line number of the the first name on the list.

Of course, we can no longer search the list by starting at the top and working down to the bottom. Now we must follow the links given by the "next" column until we find the name that we are looking for.

Now, let's examine how a new person gets put into the list. First we must find a free line in the list. Since we can delete items from the list, a free line can occur anywhere. (We will examine in detail exactly how we find a free line in a moment.) For an example we will use the list above, which has line 3 free. Now let's place a new entry for the name Mike (just pulled

it out of my hat) into the list. The name is placed on the free line; then we must place the line into the linked list.

We do this by scanning the list and finding the line containing the entry which comes alphabetically just before the one we wish to insert. This is relatively easy; we just start at the beginning of the list and compare it to the name we wish to insert. If it is before the new name (alphabeticaly) then we move on to the next name. When we find the name that comes after the new entry, the number of that line goes into the "next" column of the new entry, and the "next" column of the previous line receives the line number of the new entry.

In our example, this means that we start at Ann (the first person), go next to Jill, then to John, before we get to Sam. The name Mike will come after John and before Sam.

To perform the actual insertion, it is first necessary to get a "next person" number for the new entry; for this use the line number from the "next" column of the entry just before the new one in alphabetical order. This will cause the next person to come after the new entry, which is where we want it. Then all we need is a new "next person" number for the entry we just took the number from. This number, of course, is the line number of the new entry. The insertion is now complete.

In the example that we have been using, this means that we would move the 2 from line 5 to line 3, and put 3 in its place on line 5. The result would look like this:

Line Number	Name	Next
1	Jill	5
2	Sam	Ø
3	Mike	2
4	Ann	1
5	John	3

When looking for free lines, we simply keep the free lines in a linked list, too. We can do this by keeping a separate variable containing the number of the first free line. The first free line then has the number of the next free line in it's "next person" column. When an entry is deleted, that line number must be added to the list of free lines. The new free line becomes the first free line. To do this, the value of the variable containing the first free line is put into the new free line's "next person" column. The variable for the first free line then changes to the number of the new free line.

Obviously, this is not a method that people use in real life to solve this kind of problem, but computers can easily follow the links in the list and make it appear to the user that the names are alphabetized in the more standard manner.

Let's examine the program and see how this is done in BASIC.

```
100 REM A SIMPLE MAILING ADDRESS PROGRAM
 BY MIKE SCHULTZ
110 REM WRITTEN 8/4/1983
120 REM CRACKING THE 99/4A
130 OPTION BASE 1
140 DIM ADDR$(20,7),NEXTA(20)
150 MAXADR=20
160 MODADR$="N"
170 CALL CLEAR
180 PRINT "  THE MAILING LIST MANAGER":
: : : : : : : : : :
190 INPUT "READ IN OLD LIST FROM TAPE?":
Q$
200 IF Q$="Y" THEN 260
210 IF Q$<>"N" THEN 190
220 LSTADR=0
230 FIRSTA=0
240 FREE=0
250 GOTO 270
260 GOSUB 1510
270 CALL CLEAR
280 PRINT " THE MAILING LIST MANAGER"
290 PRINT : :" M - MODIFY THE LIST": :"
S - SEARCH THE LIST": :" Q - QUIT THE MA
NAGER": : : :"SELECT A FUNCTION:"
300 CALL KEY(0,F,S)
310 IF S=0 THEN 300
320 CALL HCHAR(23,22,F)
330 C=POS("QMS",CHR$(F),1)
340 IF C>0 THEN 370
350 CALL SOUND(500,220,0)
360 GOTO 300
370 IF C>1 THEN 410
380 IF MODADR$="N" THEN 400
390 GOSUB 1400
400 END
410 ON C-1 GOSUB 430,1620
420 GOTO 270
```

The previous lines are essentially the controlling routine described in the last article, with a few additions. Lines 230 and 240 initialize two new variables, FIRSTA and FREE. These are the variables that indicate the first lines in the lists of addresses and free lines. If there are no names on the list (a new list), then FIRSTA and FREE will have zero as their values.

One other new addition to this code is the array NEXTA on line 140. It is a numeric array that contains the next person (or address) line number for each line in array ADDR$.

```
430 REM THIS ROUTINE ACCEPTS NAMES AND A
DDRS
440 CALL CLEAR
450 PRINT "MODIFY THE MAILING ADDRESSES"
460 PRINT : :" A - ADD TO THE LIST": :"
M - MODIFY AN ADDRESS": :" D - DELETE AN
 ADDRESS": :" Q - RETURN TO MENU"
470 PRINT : : :"SELECT FUNCTION:"
480 CALL KEY(0,F,S)
490 IF S=0 THEN 480
500 CALL HCHAR(23,20,F)
510 C=POS("QAMD",CHR$(F),1)
520 IF C>0 THEN 550
530 CALL SOUND(500,220,0)
540 GOTO 470
550 IF C>1 THEN 570
560 RETURN
570 ON C-1 GOSUB 590,730,960
580 GOTO 440
```

This is the menu routine for the code that modifies names on the mailing list. It follows the same ideas described in the last section. From this menu, the user can enter commands to add a name to the list, modify an address on the list, or delete a name.

```
590 PRINT "ADD ADDRESS"
600 IF FREE=0 THEN 640
610 CURADR=FREE
620 FREE=NEXTA(FREE)
630 GOTO 690
640 IF LSTADR<MAXADR THEN 670
650 PRINT "SORRY, ADDRESS TABLE FULL"
```

```
660 RETURN
670 LSTADR=LSTADR+1
680 CURADR=LSTADR
690 GOSUB 1150
700 GOSUB 1260
710 MODADR$="Y"
720 RETURN
```

This routine handles the addition of a new name to the list. First it will find a line on which to place the name. If there are no lines available in the middle of the array (FREE=Ø), then LSTADR will be used to place the new entry on the line after the very last used row in the array. LSTADR is used to determine this row in much the same way it was used in the previous version of this program.

If there is an available line from the middle of the array, then the number of that line will be in FREE. Note in line 620 the code for obtaining the next value for FREE.

In either case, the line number of the available line is placed in the variable CURADR. Then the subroutine at line 1150 is called to get the information from the user for the new entry (except, of course, the number of the next alphabetical line). This subroutine then calls the routine at 1260 to place the name into the correct location in the linked list.

```
730 PRINT "MODIFY ADDRESS"
740 GOSUB 1800
750 IF CURADR>0 THEN 770
760 RETURN
770 PRINT "(1)NAME: ";ADDR$(CURADR,1)
780 PRINT "(2)ADDR: ";ADDR$(CURADR,2)
790 PRINT "(3)      ";ADDR$(CURADR,3)
800 PRINT "(4)CITY: ";ADDR$(CURADR,4)
810 PRINT "(5)STATE:";ADDR$(CURADR,5)
820 PRINT "(6)ZIP:  ";ADDR$(CURADR,6)
830 PRINT "(7)PHONE:";ADDR$(CURADR,7)
840 PRINT :"WHICH LINE TO CHANGE";
850 INPUT I
860 IF I<=0+I>7 THEN 940
870 INPUT "NEW LINE: ":ADDR$(CURADR,I)
880 IF I>1 THEN 950
890 IF BACK1=0 THEN 920
900 NEXTA(BACK1)=NEXTA(CURADR)
910 GOTO 930
```

```
92Ø FIRSTA=NEXTA(CURADR)
93Ø GOSUB 126Ø
94Ø MODADR$="Y"
95Ø RETURN
```

This routine allows the user to modify the information on a line. It calls the routine at 18ØØ to get from the user the name of the person whose address should be modified. This routine will search the list and return that person's line number in the variable CURADR. It will also, by the way, set the variable BACK1 to the line number of person in the list just before the line we wish to modify.

This subroutine then displays the current contents of the line, asks for the part of the line to change, and allows the user to enter the new value (lines 77Ø-87Ø).

Now all is well unless the user modifies the name, in which case it may have changed position in the list. Program line 88Ø checks for this and, if the name was changed by the user, then lines 89Ø-93Ø essentially remove the line from the linked list (without putting it on the list of available lines) and then call the routine at 126Ø again to search the list and insert the line.

```
96Ø REM THIS ROUTINE DELETES A NAME FROM
 THE LIST
97Ø PRINT "DELETE ADDRESS"
98Ø GOSUB 18ØØ
99Ø GOSUB 17ØØ
1ØØØ INPUT "DELETE (Y/N)? ":A$
1Ø1Ø IF A$="Y" THEN 1Ø4Ø
1Ø2Ø IF A$<>"N" THEN 1ØØØ
1Ø3Ø RETURN
1Ø4Ø IF BACK1=Ø THEN 1Ø7Ø
1Ø5Ø NEXTA(BACK1)=NEXTA(CURADR)
1Ø6Ø GOTO 1Ø8Ø
1Ø7Ø FIRSTA=NEXTA(CURADR)
1Ø8Ø NEXTA(CURADR)=FREE
1Ø9Ø FREE=CURADR
11ØØ FOR I=1 TO 7
1110 ADDR$(CURADR,I)=""
112Ø NEXT I
113Ø MODADR$="Y"
114Ø RETURN
```

This routine removes a name from the list. The routine at program line 1800 is called again to ask the user for the name of the person to be removed from the list. It then finds the line number they are on. The routine at 1700 is called to display the person's information so that the user can be sure of deleting the right person's entry. Finally the user is given one last chance to back out of the delete in lines 1000-1030.

If the user really does wish to delete the name and address, then the line number is placed on the free line list. After that, the information on the line is deleted (it's completely gone, it's too late now to change your mind!).

```
1150 REM THIS ROUTINE ACCEPTS A SINGLE N
AME
1160 REM AND PLACES IT IN THE ROW INDICA
TED BY "CURADR"
1170 INPUT "NAME: ":ADDR$(CURADR,1)
1180 INPUT "ADDR: ":ADDR$(CURADR,2)
1190 INPUT "ADDR(2): ":ADDR$(CURADR,3)
1200 INPUT "CITY: ":ADDR$(CURADR,4)
1210 INPUT "STATE: ":ADDR$(CURADR,5)
1220 INPUT "ZIP: ":ADDR$(CURADR,6)
1230 INPUT "PHONE: ":ADDR$(CURADR,7)
1240 RETURN
```

This routine gets the information from the user about a new address. The variable CURADR indicates on which line of the array ADDR$ the information should be placed.

```
1250 REM THIS ROUTINE LOCATES A NAME'S C
ORRECT LOCATION IN THE LIST AND PLACES I
T IN THE LIST
1260 I=FIRSTA
1270 BACK1=0
1280 IF I=0 THEN 1330
1290 IF ADDR$(CURADR,1)<ADDR$(I,1)THEN 1
330
1300 BACK1=I
1310 I=NEXTA(I)
1320 GOTO 1280
1330 IF BACK1=0 THEN 1370
1340 NEXTA(CURADR)=NEXTA(BACK1)
```

```
135Ø NEXTA(BACK1)=CURADR
136Ø RETURN
137Ø NEXTA(CURADR)=FIRSTA
138Ø FIRSTA=CURADR
139Ø RETURN
```

This routine finds the correct location within the linked list for the name on the line indicated by CURADR. It modifies the linked list to accomplish this.

It works like this: the variable I will be set to the line number of the first line on the alphabetized list. The name on line I is compared to the name contained on the new line, indicated by CURADR. The IF statement will fail if the name indicated by CURADR is alphabetically past the name indicated by I. This means that we must look farther along the list. We do so by making the variable I contain the line number of the next line in the list (program line 131Ø). But before we do, remember that, in order to do the insert, we also need to know the line number of the line before the new line in the list. So we'll save a copy of the current line into BACK1, just in case the next value of I indicates a name alphabetically after the name in the line we want to add.

When the IF statement finds the correct location in the list, BACK1 will contain the line number for the name just before the new entry in the list. Remember, the NEXTA column of the entry indicated by BACK1 saves the number of the line that follows the BACK1 entry; thus, this is the line that will follow the new entry. The insertion can now be performed. Since the NEXTA column of the line contains the number of the line to follow the new entry, we move this number to the NEXTA column of the new entry. And, since the new entry is to follow the BACK1 entry in the list, we place the number of the new line (CURADR) into the NEXTA column of the BACK1 entry and we are finished.

By now I am sure that I have given you a headache. If it is still not clear exactly what is going on, take a piece of paper and draw the process out. It is really quite simple, it is not a technique that people use everyday, so it takes some time to become comfortable with it.

The rest of the routine (program lines 128Ø, 133Ø and 137Ø-138Ø) all deal with the case when the line to be added is at the top of the list. In this case, the variable FIRST needs to be updated, instead of the line indicated by BACK1.

```
14ØØ REM THIS ROUTINE SAVES THE ADDRESS
TO FILE
1410 OPEN #1:"DSK1.CS1",SEQUENTIAL,INTER
```

```
NAL,OUTPUT,FIXED 192
142Ø PRINT #1:LSTADR,FIRSTA,FREE
143Ø FOR I=1 TO LSTADR
144Ø FOR J=1 TO 7
145Ø PRINT #1:ADDR$(I,J),
146Ø NEXT J
147Ø PRINT #1:NEXTA(I)
148Ø NEXT I
149Ø CLOSE #1
15ØØ RETURN
```

This routine is essentially the same as the save routine in the previous version of the program, except that the current values of FIRSTA and FREE also need to be saved (program line 142Ø) and the value of NEXTA for each line needs to be saved (program line 147Ø).

```
151Ø REM THIS ROUTINE READS THE ADDRESSE
S FROM FILE
152Ø OPEN #1:"DSK1.CS1",SEQUENTIAL,INTER
NAL,INPUT ,FIXED 192
153Ø INPUT #1:LSTADR,FIRSTA,FREE
154Ø FOR I=1 TO LSTADR
155Ø FOR J=1 TO 7
156Ø INPUT #1:ADDR$(I,J),
157Ø NEXT J
158Ø INPUT #1:NEXTA(I)
159Ø NEXT I
16ØØ CLOSE #1
161Ø RETURN
```

This is the routine that reads the file in from the cassette. Notice that it is essentially the reverse of the save routine.

```
162Ø REM THIS ROUTINE PRINTS THE INFORMA
TION IN ADDR$
163Ø CURADR=FIRSTA
164Ø IF CURADR=Ø THEN 168Ø
165Ø GOSUB 17ØØ
```

```
166Ø CURADR=NEXTA(CURADR)
167Ø GOTO 164Ø
168Ø INPUT "LIST COMPLETE: ":Q$
169Ø RETURN
```

This routine displays the current list to the user. It takes each line of the list, starting with the first line, calls the routine at 17ØØ to display the contents of the line, and finds the next line by getting its number from the NEXTA column of the current line.

Coming Attractions: This routine will be greatly enhanced in the next section to allow the user to find an individual name on the list.

```
17ØØ PRINT "NAME: ";ADDR$(CURADR,1)
171Ø PRINT "ADDR: ";ADDR$(CURADR,2)
172Ø IF ADDR$(CURADR,3)="" THEN 174Ø
173Ø PRINT "      ";ADDR$(CURADR,3)
174Ø PRINT "CITY: ";ADDR$(CURADR,4)
175Ø PRINT "STATE: ";ADDR$(CURADR,5);" Z
IP: ";ADDR$(CURADR,6)
176Ø PRINT "PHONE: ";ADDR$(CURADR,7)
177Ø PRINT
178Ø RETURN
```

This routine displays for the user the current contents of the line indicated by the variable CURADR. It is called by several routines (which allows the user to see the list's entries displayed in the same format evertime). It displays the line currently being processed.

```
179Ø REM THIS ROUTINE SEARCHES FOR A NAM
E AND RETURNS IT'S LOCATION IN "CURADR".
"BACK1" IS ALSO SET UP
18ØØ INPUT "NAME: ":A$
181Ø IF A$<>"" THEN 184Ø
182Ø CURADR=Ø
183Ø RETURN
184Ø BACK1=Ø
185Ø CURADR=FIRSTA
186Ø IF CURADR=Ø THEN 191Ø
187Ø IF ADDR$(CURADR,1)=A$ THEN 193Ø
```

```
188Ø BACK1=CURADR
189Ø CURADR=NEXTA(CURADR)
19ØØ GOTO 186Ø
191Ø PRINT "NAME NOT FOUND"
192Ø GOTO 18ØØ
193Ø RETURN
```

And finally, this is the routine thats searches the list to find for the user the line that matches the name provided by the user.

First it asks for the name to be found. It then starts at the first of the list looking for the name. In case the name isn't actually on the list, we check to see if we have reached the end of the list (line 186Ø). If we haven't reached the end, we check to see if it is the one we want (line 187Ø). If not, then we proceed to the next line. If it is, then we return the calling routine with the desired line number in CURADR.

If we run out of list, the name is not there and we tell the user this. We allow the user to enter the name again (lines 191Ø and 192Ø).

That is the end of the program! Next, we'll discuss how to make the routine that displays the names on the list (the search function), a bit more usable.

PART IV / USER FRIENDLINESS

This section concludes our discussion about the program that we have been using as an example, the simple mailing list program. In the previous sections we have taken this program from an idea to an actual program capable of 1) adding a person's name and address to the list, 2) deleting the person from the list, 3) modifing a person's information, 4) storing and retreiving the list to and from cassette, and 5) displaying the list to the user.

Now there are only two things left to do. We must change the routine that displays the list to the screen to instead ask for the name of a person, search the list for that person, and display the address. We must also change the routine that searches the list to allow the user to enter a guess for the person's name.

Both of these ideas fall under the concept of "User Friendliness". This concept can be loosely thought of as making a program as easy to use as possible. In this case, it is allowing a user to request the name of a particular individual directly without having to watch every name on the list

slowly scroll by. In this case, it also involves building into one of the fundamental routines of the program, the ability to allow the user to pick one name from the many that may match the user's "guess".

Unfortunately, there are very few general techniques that we can use to implement user friendly programs. But I do have a few rules that I follow: 1) Design the program so that it can be easily changed later. 2) Save as many of the user friendly features for implementing last. Get the heart of the program working first. 3) Pay attention to your (and other's) reactions to using the program. The feeling or comment that "it sure would be easier if...", is a clue to where improvement can be made, and what form it should take. As you become more experienced in programming, then you will remember these user friendly features when you write other programs and design them in from the start.

Now, onto the new routine to display addresses.

```
1620 REM THIS ROUTINE SEARCHES THE INFOR
MATION IN ADDR$
1630 CALL CLEAR
1640 PRINT TAB(8);"NAME SEARCH": :
1650 GOSUB 1800
1660 IF CURADR=0 THEN 1690
1665 PRINT
1670 GOSUB 1700
1680 GOTO 1640
1690 RETURN
```

This routine may be entered into the program from the last section by simply loading the old program using the "OLD" command (you did enter the program from last time didn't you?) and entering the above lines directly. This will replace the routine in the old program and you can see this new routine works.

This routine is quite simple, like the routine it replaces. At line 1650, the search routine located at 1800 is called to locate the name to display the address for and if that name exists then the routine at 1700 is called to perform the actual display.

Now enter the lines below to replace the routine at 1800 that searches the list and see now it functions.

To the user of the program this routine has the following appearence: The user enters the name they are looking for. This routine finds all the names in the list that match the name entered and displays them. The user then is requested exactly which one of the displayed names is desired and location of that name is returned to the calling routine.

Please note that this routine, when it is searching for matches, it is looking at only the first part of the names in the list and comparing only as many characters as in the orginal name to them. This means that if we want to pick a name from the D's we enter a "D", and if we want to pick from the David's we enter "DAVID".

```
1790 REM THIS ROUTINE SEARCHES FOR A NAM
E AND RETURNS ITS LOCATION IN "CURADR".
"BACK1" IS ALSO SET UP
1800 INPUT "NAME: ":A$
1810 IF A$<>"" THEN 1840
1820 CURADR=0
1830 RETURN
1840 BACK1=0
1850 CURADR=FIRSTA
1860 L=LEN(A$)
```

This part of the routine sets up for the search. Note line 1860 which finds the length of the name that we are looking for.

```
1870 IF CURADR=0 THEN 2160
1880 IF SEG$(ADDR$(CURADR,1),1,L)>=A$ TH
EN 1920
1890 BACK1=CURADR
1900 CURADR=NEXTA(CURADR)
1910 GOTO 1870
```

This part of the routine searches the list for the first name that matches. Notice the SEG$ string command which is used to isolate only that part of the name on the list that we wish to look at.

```
1920 J=CURADR
1930 I=0
1940 C=1
1950 IF CURADR=0 THEN 2040
1960 IF SEG$(ADDR$(CURADR,1),1,L)>A$ THE
N 2040
1970 I=I+1
```

```
1980 PRINT I;ADDR$(CURADR,1)
1990 CURADR=NEXTA(CURADR)
2000 C=C+1
2010 IF C<=22 THEN 1950
2020 INPUT "ENTER # OF DESIRED NAME OR
0 TO CONTINUE: ":N
2030 IF N<=0 THEN 1940 ELSE 2070
2040 IF I=0 THEN 2160
2050 PRINT "END OF LIST"
2060 INPUT "ENTER # OF DESIRED NAME: ":N
2070 IF (N<0)+(N>I)THEN 2060
2080 IF N=0 THEN 2160
```

This routine displays all the names in the list that match the desired name. Note that variable I is used to keep a count of the matched names. It is displayed with each name so that later can give the number of the desired name. Note that variable C is used to keep a count of the number of names currently displayed on the screen. When the screen is full, this part of the routine pauses to allow the user time to examine the screen or to select the desired name, if it is already visible.

```
2090 CURADR=J
2100 IF N=1 THEN 2150
2110 BACK1=CURADR
2120 CURADR=NEXTA(CURADR)
2130 N=N-1
2140 GOTO 2100
2150 RETURN
```

This part of the routine is entered whenever the user has selected the number of the desired name. This part resets CURADR to the location of the name that matched. Then, CURADR is zipped down the list the desired number of names and control is returned to the calling routine.

```
2160 INPUT "NAME NOT FOUND":A$
2170 CURADR=0
2180 BACK1=0
2190 RETURN
```

This part of the routine is entered when either the routine or the user can't find the desired name. The routine admits failure and returns.

And that concludes this tutorial of BASIC programming dealing with the mailing program as an example.

Happy computing !

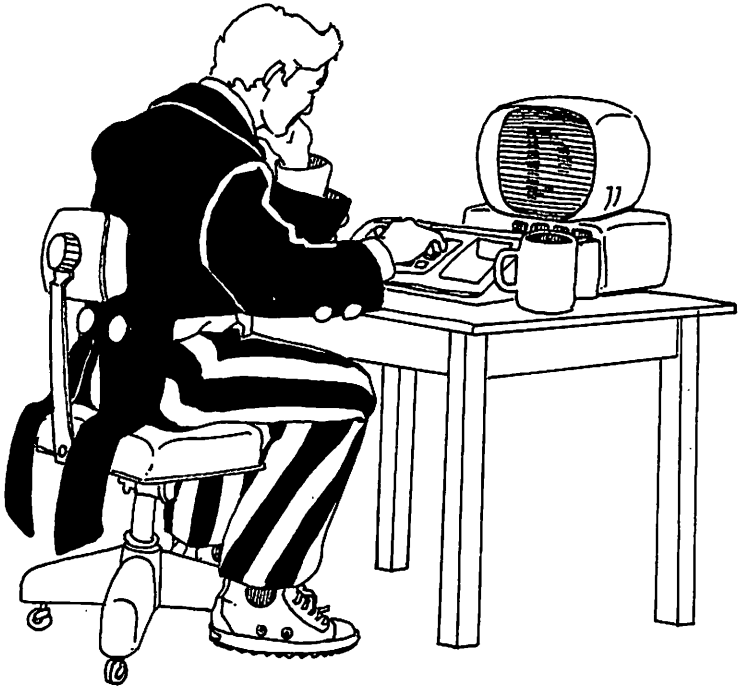
LOGICAL OPERATORS

Brian Prothro

PART I

Learning to use logical operators is a lot like learning to juggle. At first it's total confusion, but once you master the task it's quite an impressive trick. Attempts to encode logical algorithms in Basic, without understanding logical operators, can create cumbersome code with with lengthly calculation times. This tutorial provides a few examples on how to reduce your code to a compact logical algorithm. You should have a familiar understanding of Basic before you attempt this tutorial. If you are very familiar, hang in there, we'll soon get off the ground.

Unfortunately the most clearly understandable format for logical operators (AND,OR,NOT and XOR) is available only with extended Basic. However regular Basiccan accomplish the same task using the mathmatical symbols (+) plus, (-) minus, (*) multiply, and (=) equals. First we will learn simple theory using extended Basic, it is easier to read, then we will duplicate the code in regular Basic.



Here are the definitions for two of the four operators.

AND - The statement is TRUE if, and only if both expressions are TRUE. AND means both.

OR - The statement is TRUE if one expression, or both expressions are TRUE. OR means one or both.

What do we mean by this? Try applying the OR definition to the coded example below.

OR example: IF (A=10) OR (B=20) THEN PRINT "SUCCESS"

Reads like English, right! You are testing for the truth of two expressions. If A=10 and/or if B=20. By the OR definition, if one of the expressions is TRUE, or if both expressions are TRUE then the whole statement is TRUE, and the word SUCCESS will be printed by the computer. OR means one or both. The only other possible result: If and only if both expressions in the statement are FALSE will the computer not print SUCCESS.

You have probably seen that the (IF . . . THEN . . .)statement is central to the logical process. If the result of your logical comparison after the IF is TRUE, then your code after THEN will execute.

AND example: IF (A=10) AND (B=20) THEN PRINT "SUCCESS"

With the AND statement both expression must be TRUE for the statement to be TRUE. If for instance, the variable (A) in the above statement were 5 when the expression asks for it to be equal to 10, then this expression in our statement would be FALSE, the statement would not print SUCCESS. The entire statement would not be TRUE because both expressions are not TRUE. Simple !

Here are two AND-OR truth tables. With these tables as a reference, you can write or decifer logical statements with two expressions. The tables are read horizontally, with the results of the statement given. See how the tables reflect the above definitions. Try each one mentally.

AND TABLE

(A)	(B)	LOGICAL RESULT
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

OR TABLE

(A)	(B)	LOGICAL RESULT
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

All possible results for two expresions can be deduced from the above table. Lets pick up the pace.

PART II

In a given program, you may eventually ask for a users input. Lets say you only want this input to be within certain numerical limits. For example, the number 2 and all numbers between 10 and 15. Examine the program code below and see how the problem is solved.

```
100 PRINT "INPUT YOUR NUMBER"
200 INPUT X
300 IF X=2 OR (X>=10 AND X<=15) THEN PRINT "ACCEPTED" ELSE 100
400 END
```

With the combined logical operators, we can see that if the users entry is not 2, or a number from 10 to 15, then program control returns and asks that the number be entered again. In this way you can screen a users input for correct entries.

Before we continue, let's understand what the computer does when it interprets a logical operator. The computer weighs a statement such as A=10, if this expression is TRUE then 99/4A logic assigns a value of (-1) to the expression. If the expression is FALSE, a zero (0) is assigned. The value of this information is revealed when we use the regular Basic version of logical operators. Although the extended Basic version is easier to understand, the regular version is more revealing, and is mathmatically closer to reality. It's all simple math !

These examples show two EQUIVALENT expressions.

```
EXTENDED:  IF A=10 AND B=20 THEN PRINT "SUCCESS"
 REGULAR:  IF (A=10)*(B=20)<>0 THEN PRINT "SUCCESS"
```

To see how this regular Basic statement is computed, lets assume that both experssions for A and B are TRUE (-1), making the statement TRUE(-1). Since the computer has assigned a value of (-1) to each expression, the result would look like this:

```
REGULAR:  IF (-1)*(-1)<>0 THEN PRINT "SUCCESS"
```

Since (-1)X(-1) = 1, one is not equal to zero, as the statement requires. So our equation/statement is TRUE, and the word SUCCESS will be printed by the computer. To make a long lesson short, examine how the FALSE outcome below is obtained.

```
IF (0)*(-1)<>0 THEN PRINT "SUCCESS"
```

(0)X(-1) = 0, since zero is not greater or less than (0), as the statement requires, the statement is FALSE and SUCCESS will not be printed.

Let me now give you some standard formats for coding regular Basic statements, although they can be written any number of different ways.

AND - Multiply the two expressions and check if greater or less than zero.

```
EXTENDED:  IF A=10 AND B=10 THEN . . .
  REGULAR:  IF (A=10)*(B=10)<>0 THEN . . .
```

OR - Add the two expressions and check if less than zero.

```
EXTENDED:  IF A=10 OR B=20 THEN . . .
  REGULAR:  IF (A=10)+(B=20)<0 THEN . . .
```

The logical charts we examined before may now be viewed in a different light, a mathmatical one. They are, of course, equivalent to the AND/OR charts shown previously.

AND logic chart

(A)	(B) =	LOGICAL RESULT
-1	-1	-1
-1	0	0
0	-1	0
0	0	0

OR logic chart

(A)	(B) =	LOGICAL RESULT
-1	-1	-1
-1	0	-1
0	-1	-1
0	0	0

At this point you should put this book down and physically try some examples before you go on. You often see this suggested, but in this case it's good advice.

PART III

I will outline NOT and XOR briefly. With the logical charts on NOT and XOR you should be able to deduce what you need when using them. These last two of the four operators reverse our thinking a bit.

DEFINITIONS:

XOR - The statement is TRUE if, and only if, ONE of the expressions are TRUE, but not if both are TRUE. XOR means either one, but not both.

NOT - Is used in conjunction with other operators to reverse a logical outcome.

NOT LOGIC CHART

(A)	NOT(A)
TRUE	FALSE
FALSE	TRUE

NOT simply reverses logic, (0) becomes (-1), and (-1) becomes (0). NOT can also reverse the logic of an entire statement. It is used where needed.

XOR LOGIC

(A)	(B)	OUTCOME
-1	-1	0
-1	0	-1
0	-1	-1
0	0	0

XOR LOGIC CHART

(A)	(B)	RESULT
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Get ready, these are equivalent statements for XOR:

EXTENDED: IF A=2 XOR B=4 THEN . . .

REGULAR: IF -(((A=2 + B=4)<0)*((A=2 * B=4)=0))< >0 THEN . . .

Given this XOR equation, we can reduce it step by step. First the computer reduces an equation to its simplest form, then depending on if the expression is TRUE or FALSE, it assigns a (-1) or (0) to the expressions. We now will assume, for this example, that A is not equal to TWO (FALSE), and that B is equal to FOUR (TRUE). The equation appears like this, and the final result will be TRUE.

```
      IF-(((A=2   + B=4)<0) * ((A=2  *  B=4)=0))<>0  THEN . . .

1)    -((( 0    +  -1 )<0) * (( 0   *   -1 )=0))<>0  THEN . . .

2)    -((       -1       <0) * (      0       =0))<>0  THEN . . .

3)    -((          -1        ) * (      -1          ))<>0  THEN . . .

4)    -(                    1                    )<>0

5)                          -1                     <>0

6)                                  -1
```

Note that most of the steps are purely math, but in steps three and six the computer is assigning logical values respective to the validity (TRUTH) of each our statements.

XOR logic states that if only one expression is TRUE then (as is the case above), the statement is TRUE (-1). As you can tell, it's easier to look at the logic table and write your code than to figure it out from scratch. Given the program results you require, and one or more of the logical tables/operators, you can solve any problem.

NOT simply reverses the logic on an expression or on an entire state ment. It is used as needed. If expression (B) is TRUE then NOT(B) woul d be FALSE. The code would look like this:

```
IF A=2 AND NOT B=4 THEN . . .
```

For example, if you wanted all results of the AND table to be reversed, then you would write your code, as shown below, with the NOT operator leading your statement.

```
IF NOT (A=2 AND B=4) THEN . . .
```

NOT simply reverses the logic. (0) becomes (-1) and (-1) becomes (0).

PART IV

Logical operations can be used to compare filenames,character strings, or numbers, or to identify a particular segment of text. This example is valid. It checks to see if the ASCII characters in variable A$ are equivalent to a segment of the string B$, or if A$ equals the string "KLMN".

```
IF A$=SEG$(B$,4,LEN(B$)) OR A$="KLMN" THEN . . .
```

For example, we could use a call key statement in a program. A users input could be screened so that only the ASCII characters 38, 40 and 45 through 48 are accepted.

```
100 CALL KEY(A)
200 IF A=38 OR A=40 OR (A>=45 AND A<=48) THEN 300 ELSE 100
300 PRINT "CHARACTER ENTRY ACCEPTED"
```

ORDER OF PRECIDENCE

When the computer executes a series of expressions containing logical operators, it executes them in a specific order. The order of precidence (first to last) is NOT, XOR, AND and OR. So if you have several of these in a statement, the order of execution will not necessarily be from left to right. If you are not careful, your code may kick up some unexpected results.

The first statement is an example. The second statement contains parenthesis and indicates the computers order of preference for the example. Innermost parenthesis are always executed first.

```
WE SEE:   IF A=10 AND B=17 AND D=Y OR F=27 AND NOT E=12 THEN . . .

COMPUTER: IF ((A=10 AND B=17) AND D=Y) OR (F=27 AND (NOT E=12)) THEN . . .
```

NOT is executed first, so it is shown in the innermost parenthesis. Next, expressions A=10, B=17, and D=Y are reduced to a logical TRUE or FALSE. Then expressions F=27, and E=12. Last, the remaining values on both sides of the OR operator are reduced, since OR is last in the order of preference. Well see this type of solution in more detail with the next example, don't fret.

If this order did not give us the results our program required, we could force the order of execution by including our own parenthesis. Lets do that using the same expression. I will choose an arbitraray order by including parenthesis.

```
EXT.  IF A=10 AND ((B=17 AND D=Y) OR (F=27)) AND NOT E=12 THEN . . .
```

Here is what the solution would look like given the following conditions.

A=10, FALSE B=17, TRUE D=Y,TRUE F=27, FALSE E=12, TRUE

IF A=10 AND ((B=17 AND D=Y) OR (F=27)) AND NOT E=12 THEN . . .

LOGICAL EQUIVALENT: Remember this is a logical solution only, not math.

```
    IF 0 AND ((-1   AND  -1) OR  (0)) AND - (-1) THEN . . .
 1) IF 0 AND ((-1   AND  -1) OR  (0)) AND   0  THEN . . .
 2) IF 0 AND ((-1          ) OR   0 ) AND   0  THEN . . .
 3) IF 0 AND (              -1        ) AND   0  THEN . . .
 4)  IF            0                    AND   0  THEN . . .
 5)  IF                              0           THEN . . .
```

1). The order starts with E=12 having its logic reversed by the NOT operator. Its TRUE becomes FALSE.
2). Next, the expressions B=17 and D=Y. They are both TRUE so this AND comaprison results in TRUE.
3). Then this same result would be compared with F=27 by the OR operator. F=27 is FALSE but in an OR statement only one expression is required to be TRUE, so this results in TRUE. Now there remains three expressions each seperated by AND's.
4). The first two, FALSE and TRUE result in FALSE (0).
5). Then this result is compared with the last value of . Again, since both expressions are not TRUE, the final result is FALSE. Since the final result of our statement is FALSE (0), our task will not execute.

I will finish this section with something simple. these two statements are equivalent.

```
EXT.  IF B>10 AND B<20 THEN PRINT "SUCCESS"
REG.  IF 10<B<20 THEN PRINT "SUCCESS"
```

As you can deduce it is the ability of AND, OR, NOT & XOR to combine in any number of ways that allows you to compare and check an incredible number of conditions in one compact statement. (If you so desire.) Like I said, it's like learning to juggle. When writing your code it requires thought to ensure that all possible combinations of your logical expressions will result in the proper execution of your code. If you find that you have trouble writing a statement, make a loop that allows you to keep entering different values for your variables. Then watch to see what your PRINT statement displays.

As you can see the flexibility of logical expressions can be a trying experience, but the effort is well worth the trouble in execution time and reduction of program code.

Now I'll describe two logical statements from an example program. The only thing this program does is allow you to input the upper,lower,left, and right boundries within which a cursor will have limited movement. The first statement we will look at validates your input. The second statement is somewhat more complex.

```
100 REM CRACKING THE 99/4A
110 REM LOGIC DEMO #1
120 REM BRUCE WYCHE 02/08/84
130 CALL CLEAR
140 CALL SCREEN(2)
150 PRINT TAB(7);"LOGIC DEMO PROGRAM":TA
B(7);"CRACKING THE 99/4A"
160 FOR I=1 TO 11
170 PRINT
180 NEXT I
190 PRINT " PRESS ANY KEY TO CONTINUE"
200 CALL SCREEN(15)
210 CALL KEY(3,K,S)
220 IF S=0 THEN 210
230 CALL CLEAR
240 INPUT "  ENTER LEFT BORDER COLUMN #
    (3 <= LB < 29) ":LT
250 IF (LT>2)*(LT<29)>0 THEN 280
260 PRINT "ERROR, TRY AGAIN"
270 GOTO 240
280 INPUT "  ENTER RIGHT BORDER COL #
    ("&STR$(LT)&" < RB <=30) ":RT
290 IF (RT>LT)*(RT<31)>0 THEN 320
300 PRINT "ERROR, TRY AGAIN"
310 GOTO 280
320 INPUT "  ENTER TOP  BORDER   ROW  #
    (1 <= TP <=23) ":TP
330 IF (TP>0)*(TP<24)>0 THEN 360
340 PRINT "ERROR, TRY AGAIN"
350 GOTO 320
360 INPUT "  ENTER BOTTOM BORDER ROW #
    ("&STR$(TP)&" < BM <=24) ":BM
370 IF (BM>TP)*(BM<25)>0 THEN 400
```

```
380 PRINT "ERROR, TRY AGAIN"
390 GOTO 360
400 CALL CLEAR
410 CALL HCHAR(TP,LT,30,RT-LT+1)
420 CALL HCHAR(BM,LT,30,RT-LT+1)
430 CALL VCHAR(TP+1,LT,30,BM-TP-1)
440 CALL VCHAR(TP+1,RT,30,BM-TP-1)
450 R=TP
460 C=LT
470 CALL GCHAR(R,C,GC)
480 CALL HCHAR(R,C,30)
490 CALL KEY(0,K,S)
500 CALL HCHAR(R,C,GC)
510 IF S=0 THEN 480
520 REM COMPACT EXAMPLE OF           LOGI
CAL OPERATORS
530 REM HCHAR WILL LEAVE ONE         OF T
HREE CHOICES
540 REM DEPENDING ON WHICH           KEY
WAS PRESSED
550 REM LEAVE OLD CHAR 'GC'          IF K
EY WAS ARROW,           'ENTER' OR 'Fctn
 6'
560 REM LEAVE ' ' IS KEY WAS         NON-
PRINTABLE CHAR
570 REM LEAVE NEW KEY PRESS          AND
MOVE RIGHT IF           KEY BETWEEN 32,
127
580 CALL HCHAR(R,C,GC*(K>7)*(K<14)+32*(K
>131)*(K<153)+K*(K>31)*(K<128))
590 IF K<>12 THEN 610
600 GOTO 450
610 IF K<>13 THEN 650
620 R=R-(R<BM)
630 C=LT
640 GOTO 470
650 R=R+(K=10)*(R<BM)-(K=11)*(R>TP)
660 C=C-(K=8)*(C>LT)-((K>31)*(K<128)-(K=
9))*(C<RT)
670 GOTO 470
680 REM ADD YOUR OWN PROGRAM
690 REM CODE FROM HERE, DOWN
700 REM YOU ARE ON YOUR OWN.
710 END
```

Your left limit is entered in line 240. The input must be within the boundries of the screen and is checked by line 250. It uses AND logic, so both clauses must be true. This expression checks to see that the left boundry is greater than 2 and less than 29.

```
IF (LT>2)*(LT<29)<>0 THEN 280
```

When both conditions are true your entry is accepted and line 280, the next entry, is executed. In this case the above expression would look like this;

```
IF (-1)*(-1)>0 THEN 280
   true true
```

If either condition is false the returned value is zero so line 260 would print an error message and reask you to enter your input.

You can see that the statements in lines 290,330, and 370 are similar for the other boundries.

Line 580 shows three tests. CALL HCHAR will leave the old character "GC" if the key pressed was an arrow, enter, or function 6. It leaves a space if the key pressed was a non-printable character. It leaves the new key pressed and moves right if the key was between ASCII 32 or 127.

Note: 'GC' is the ASCII code for the original character in that row & col.

Line 580.

```
CALL HCHAR(ROW,COL,GC*(KEY>7)*(KEY<14)
  + 32*(KEY>131)*(KEY<153)
  + KEY*(KEY>31)*(KEY<128)
```

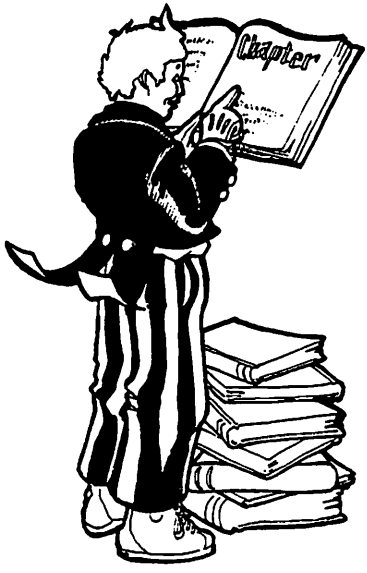
The code above is shown broken into it's three logical tests. The character printed by 'HCHAR' will be the sum of the three tests, each seperated by "+".
First test; If 'KEY' was an arrow key (ASCII value 8,9,10 or 11), then you get 'GC' as the result. The other SUMS are each zero and 'fall-out' of the equation.

Second test; If 'key' was not printable : (ASCII 131 -> 153), then leave a space.

Third test; If 'KEY' was a printable character (ASCII 32 -> 127), then display it.
Note in this example that each logical test range must not overlap.

This short example is loaded with other logical expressions. You might try a few on your own to increase your familiarity with this type of code. Good luck!

GAMES



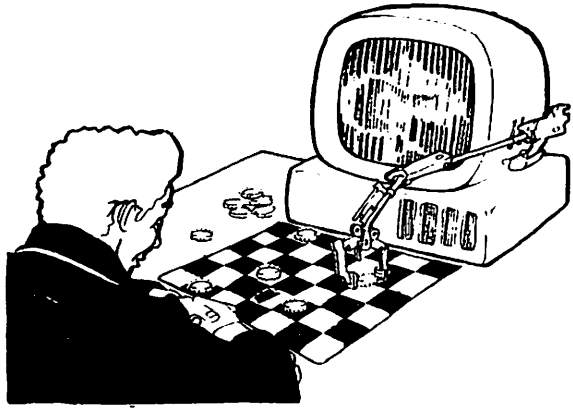
CHECKERS FOR 16K

Almost everyone has enjoyed a game of checkers at one time or another, so it is inevitable that some version of checkers would eventually make it into your computer. This version of checkers fits into 16K of console memory, while offering a fast game for beginners. The moves are timed and the time allowed for a player to make a move is decided by the level of play chosen. Level one allows a player 20 seconds to move whereas level nine allows 180 seconds.

The computer displays the game board and clock, crowns kings, and as you can guess, takes advantage of any multiple jumps left by the human challenger! By removing the introduction or using the 32K memory expansion, one could add a few subroutines that would make this program quite a contender.

All the rules are followed, including forcing the challenger to make any available jumps. Good luck!

```
100 REM CHECKERS.TO RESIGN ENTER 'FCTN 8
110 REM ALLEN HOLLEY
120 REM CRACKING THE 99/4A
130 CALL CLEAR :: CALL HEADER
140 DIM Z(8,8)
```



```
150 DATA 1,0,1,0,0,0,-1,0,0,1,0,0,0,-1,0
,-1,9
160 CALL CHAR(112,"000000000070F0F0F0F0F0
F0700000000F0F0F0E0000000000000000E0F0F
0F0"):: CALL CHAR(116,"0000000000000000"
)
170 CALL CHAR(117,"0F0F0F0F0F0F0F0FF0F0F
0F0F0F0F0F0F0")
180 CALL CHAR(120,"000000000070F0F0F0F0F0
F0700000000F0F0F0E0000000000000000E0F0F
0F0"):: CALL CHAR(124,"FFFFFFFFFFFFFFFF"
)
190 CALL CHAR(125,"0F0F0F0F0F0F0F0FF0F0F
0F0F0F0F0F0F0")
200 CALL CLEAR :: CALL SCREEN(8):: CALL
COLOR(11,2,9,12,16,9)
210 CALL HCHAR(5,1,124,32*16):: CALL VCH
AR(1,1,32,24*14):: CALL VCHAR(1,31,32,48
)
220 Y=8 :: FOR X=5 TO 19 STEP 2 :: CALL
HCHAR(X,13,48+Y):: Y=Y-1 :: NEXT X
230 Y=1 :: FOR X=15 TO 29 STEP 2 :: CALL
 HCHAR(22,X,64+Y):: Y=Y+1 :: NEXT X
240 FOR X=1 TO 8 :: FOR Y=1 TO 8 :: READ
 J :: IF J>2 THEN 250 ELSE Z(Y,X)=J :: G
OTO 260
250 RESTORE :: READ Z(Y,X)
260 NEXT Y :: NEXT X :: YOU,ME=12
270 FOR X=1 TO 8 :: FOR Y=1 TO 8 :: GOSU
B 1630 :: NEXT Y :: NEXT X
280 DISPLAY AT(2,1):"ENTER LEVEL OF PLAY
" :: DISPLAY AT(3,1):"   ENTER 1 THRU 9
 1"
290 ON WARNING NEXT :: ACCEPT AT(3,20)VA
LIDATE(DIGIT)SIZE(-1)BEEP:LIM :: LIM=LIM
*20 :: TOUT=0
300 DISPLAY AT(1,1):"YOU HAVE";LIM;"SECO
NDS" :: DISPLAY AT(2,1):" TO ENTER EACH
MOVE" :: DISPLAY AT(11,1)SIZE(6):"  TIME
"
310 REM  MY FIRST MOVE
320 A=6 :: RANDOMIZE :: CALL PEEK(-31880
,B):: B=INT(INT(B/25+1)*2)
330 C=5 :: RANDOMIZE :: CALL PEEK(-31880
,D):: D=INT(D/50):: D=B+1-D-D :: IF D=9
```

```
THEN D=7
340 REM MOVE MY MAN
350 GOSUB 1610 :: IF C=1 AND Z(A,B)=-1 T
HEN Z(A,B)=-2 :: CALL SOUND(500,500,1)::
 RJ1=0
360 Z(C,D)=Z(A,B):: Z(A,B)=0 :: X=A :: Y
=B :: GOSUB 1630 :: X=C :: Y=D :: GOSUB
1630 :: IF RJ1=0 THEN 380
370 A=C :: B=D :: GOSUB 1470 :: IF RJ1>0
 THEN 450 ELSE IF Z(A,B)=-2 THEN GOSUB 1
330 :: IF RJ1>0 THEN 450
380 DISPLAY AT(3,1):" " :: GOTO 770
390 REM MY MOVE
400 DISPLAY AT(1,1):" " :: DISPLAY AT(2,
1):" " :: DISPLAY AT(5,1)SIZE(9):" " ::
DISPLAY AT(3,1)SIZE(9)BEEP:"MY MOVE"
410 REM CK FOR JUMPS
420 OPP=1 :: FOR A=8 TO 1 STEP -1 :: FOR
 B=1 TO 8 :: IF Z(A,B)<0 THEN GOSUB 1470
 :: IF RJ1>0 THEN 450
430 IF Z(A,B)=-2 THEN GOSUB 1330 :: IF R
J1>0 THEN 450
440 NEXT B :: NEXT A :: GOTO 580
450 SJ1=RJ1 :: SJ2=RJ2 :: SJ3=RJ3 :: SJ4
=RJ4
460 FOR A=8 TO 1 STEP -1 :: FOR B=1 TO 8
 :: IF Z(A,B)<0 THEN GOSUB 1470 :: IF RJ
1>0 THEN 490
470 IF Z(A,B)=-2 THEN GOSUB 1330 :: IF R
J1>0 THEN 490
480 NEXT B :: NEXT A :: GOTO 500
490 GOSUB 1040 :: IF RJ5>0 THEN 550 ELSE
 480
500 FOR A=8 TO 1 STEP -1 :: FOR B=1 TO 8
 :: IF Z(A,B)<0 THEN GOSUB 1540 :: IF RJ
10 THEN 32767
510 IF Z(A,B)=-2 THEN GOSUB 1400 :: IF R
J1>0 THEN 530
520 NEXT B :: NEXT A :: GOTO 540
530 GOSUB 1040 :: IF RJ5>0 THEN 550 ELSE
 520
540 RJ1=SJ1 :: RJ2=SJ2 :: RJ3=SJ3 :: RJ4
=SJ4
550 A=RJ1 :: B=RJ2 :: C=RJ3 :: D=RJ4 ::
IF A<C THEN A1=A+1 ELSE A1=A-1
```

```
560 IF B<D THEN A2=B+1 ELSE A2=B-1
570 Z(A1,A2)=0 :: X=A1 :: Y=A2 :: GOSUB
1630 :: YOU=YOU-1 :: IF YOU<1 THEN 1720
ELSE 340
580 REM NO JUMPS CK FOR MOVE
590 OPP=-1 :: FOR A=1 TO 8 :: FOR B=1 TO
 8
600 IF Z(A,B)>0 THEN GOSUB 1330 :: IF RJ
1>0 THEN 620 ELSE IF Z(A,B)=2 THEN GOSUB
 1470 :: IF RJ1>0 THEN 620
610 NEXT B :: NEXT A :: GOTO 630
620 SB=B :: SA=A :: GOSUB 1260 :: IF NJ=
0 THEN RJ1=0 :: GOTO 340 ELSE B=SB :: A=
SA :: GOTO 610
630 RJ1=0 :: B=2 :: FOR A=8 TO 4 STEP -2
 :: IF Z(A,B)<>-1 THEN 640 ELSE IF Z(A-1
,B-1)=0 THEN C=A-1 :: D=B-1 :: GOTO 340
640 NEXT A
650 B=7 :: FOR A=7 TO 3 STEP -2 :: IF Z(
A,B)<>-1 THEN 660 ELSE IF Z(A-1,B+1)=0 T
HEN C=A-1 :: D=B+1 :: GOTO 340
660 NEXT A
670 A=2 :: FOR B=1 TO 8 :: IF Z(A,B)=-1
THEN GOSUB 1120 :: IF C>0 THEN 340
680 NEXT B
690 FOR A=8 TO 3 STEP -1 :: FOR B=1 TO 8
 :: IF Z(A,B)>-1 THEN 710 ELSE GOSUB 112
0 :: IF C=0 THEN 710
700 GOSUB 1160 :: IF NJ=0 THEN 340
710 NEXT B :: NEXT A
720 FOR A=8 TO 3 STEP -1 :: FOR B=1 TO 8
 :: IF Z(A,B)<0 THEN GOSUB 1120 :: IF C>
0 THEN 340
730 NEXT B :: NEXT A
740 ME=0 :: GOTO 1720
750 REM SMALL DELAY
760 FOR DEL=1 TO 20 :: NEXT DEL :: RETUR
N
770 REM HIS MOVE
780 TIME=LIM :: OPP=-1 :: FOR A=1 TO 8 :
: FOR B=1 TO 8
790 IF Z(A,B)>0 THEN GOSUB 1330 :: IF RJ
1>0 THEN 810 ELSE IF Z(A,B)=2 THEN GOSUB
 1470 :: IF RJ1>0 THEN 810
800 NEXT B :: NEXT A
```

```
810 DISPLAY AT(22,1)SIZE(9):" " :: DISPL
AY AT(20,1)SIZE(9):" " :: DISPLAY AT(20,
1)SIZE(9)BEEP:"YOUR MOVE"
820 DISPLAY AT(13,1)SIZE(9):TIME :: TIME
=TIME-.119
830 CALL KEY(3,B1,STAT):: IF TIME<0 THEN
 1720 ELSE IF STAT<1 THEN 820 ELSE GOSUB
 760 :: IF B1=6 THEN 1720
840 IF B1<65 OR B1>72 THEN 810 ELSE B=B1
-64 :: CALL HCHAR(22,3,B1)
850 CALL KEY(3,A1,STAT):: IF STAT<1 THEN
 850 ELSE GOSUB 760 :: IF A1<49 OR A1>56
 THEN 810 ELSE A=A1-48 :: CALL HCHAR(22,
5,A1)
860 CALL KEY(3,D1,STAT):: IF STAT<1 THEN
 860 ELSE GOSUB 760 :: IF D1<65 OR D1>72
 THEN 810 ELSE D=D1-64 :: CALL HCHAR(22,
8,D1)
870 CALL KEY(3,C1,STAT):: IF STAT<1 THEN
 870 ELSE GOSUB 760 :: IF C1<49 OR C1>56
 THEN 810 ELSE C=C1-48 :: CALL HCHAR(22,
10,C1)
880 IF A+2=C OR A-2=C OR B+2=D OR B-2=D
THEN 970 ELSE IF Z(C,D)<>0 OR Z(A,B)<1 T
HEN 810
890 REM IF HE MOVES
900 IF RJ1>0 THEN 810 ELSE IF A+1=C THEN
 910 ELSE IF A-1<>C THEN 810 ELSE IF Z(A
,B)<>2 THEN 810
910 IF B+1=D OR B-1=D THEN 920 ELSE 810
920 IF C<8 OR Z(A,B)=2 THEN 930 ELSE CAL
L SOUND(500,400,1):: Z(A,B)=2 :: RJ1=0
930 Z(C,D)=Z(A,B):: Z(A,B)=0 :: X=A :: Y
=B :: GOSUB 1630 :: X=C :: Y=D :: GOSUB
1630 :: IF RJ1=0 THEN 960
940 OPP=-1 :: A=C :: B=D :: GOSUB 1330 :
: IF RJ1>0 THEN 810
950 IF Z(A,B)=2 THEN GOSUB 1470 :: IF RJ
1>0 THEN 810
960 DISPLAY AT(20,1)SIZE(9):" " :: GOTO
390
970 REM IF HE JUMPS
980 IF A+2=C THEN 990 ELSE IF A-2<>C OR
Z(A,B)<>2 THEN 810
990 IF B+2=D OR B-2=D THEN 1000 ELSE 810
```

```
1000 IF Z(C,D)<>0 OR Z(A,B)<1 THEN 810 E
LSE IF C>A THEN A4=A+1 ELSE A4=A-1
1010 IF D>B THEN A5=B+1 ELSE A5=B-1
1020 IF Z(A4,A5)>-1 THEN 810
1030 Z(A4,A5)=0 :: X=A4 :: Y=A5 :: GOSUB
 1630 :: ME=ME-1 :: IF ME=0 THEN 1720 EL
SE 920
1040 REM CHECK FOR DBL JUMP
1050 RJ5=0 :: IF RJ3-2<1 THEN 1080 ELSE
IF RJ4+2>8 THEN 1070
1060 IF Z(RJ3-1,RJ4+1)>0 AND Z(RJ3-2,RJ4
+2)=0 THEN RJ5=1 :: GOTO 1110
1070 IF RJ4-2<1 THEN 1080 ELSE IF Z(RJ3-
1,RJ4-1)>0 AND Z(RJ3-2,RJ4-2)=0 THEN RJ5
=1 :: GOTO 1110
1080 IF Z(RJ1,RJ2)=-1 THEN 1110 ELSE IF
RJ3+2>8 THEN 1110 ELSE IF RJ4+2>8 THEN 1
100
1090 IF Z(RJ3+1,RJ4+1)>0 AND Z(RJ3+2,RJ4
+2)=0 THEN RJ5=1 :: GOTO 1110
1100 IF RJ4-2<1 THEN 1110 ELSE IF Z(RJ3+
1,RJ4-1)>0 AND Z(RJ3+2,RJ4-2)=0 THEN RJ5
=1
1110 RETURN
1120 REM CHECK FOR MOVE
1130 C=0 :: IF B-1<1 THEN 1140 ELSE IF Z
(A-1,B-1)=0 THEN C=A-1 :: D=B-1 :: RETUR
N
1140 IF B+1>8 THEN RETURN ELSE IF Z(A-1,
B+1)=0 THEN C=A-1 :: D=B+1
1150 RETURN
1160 REM CAN HE JUMP ME IF I MOVE
1170 NJ=0 :: IF D>7 THEN 1180 ELSE IF Z(
C-1,D+1)>0 THEN 1250
1180 IF D<2 THEN 1190 ELSE IF Z(C-1,D-1)
>0 THEN 1250
1190 IF B>6 THEN 1200 ELSE IF Z(A-1,B+1)
<0 AND Z(A-2,B+2)>0 THEN 1250
1200 IF B<3 THEN 1210 ELSE IF Z(A-1,B-1)
<0 AND Z(A-2,B-2)>0 THEN 1250
1210 IF B>6 THEN 1230 ELSE IF A<3 THEN 1
220 ELSE IF Z(A-1,B+1)<0 AND Z(A-2,B+2)>
0 THEN 1250
1220 IF A>6 THEN 1230 ELSE IF Z(A+1,B+1)
<0 AND Z(A+2,B+2)>1 THEN 1250 ELSE IF B<
```

```
3 THEN RETURN ELSE IF Z(A+2,B-2)>1 THEN
1250
1230 IF B<3 THEN RETURN ELSE IF A<3 THEN
 1240 ELSE IF Z(A-1,B-1)<0 AND Z(A-2,B-2
)>0 THEN 1250
1240 IF A>6 THEN RETURN ELSE IF Z(A+1,B-
1)<0 AND Z(A+2,B-2)>1 THEN 1250 ELSE RET
URN
1250 NJ=1 :: RETURN
1260 REM HE CAN JUMP ME
1270 IF RJ3>RJ1 THEN A=RJ3-1 ELSE A=RJ1-
1
1280 IF RJ4>RJ2 THEN B=RJ4-1 ELSE B=RJ2-
1
1290 IF Z(A-1,B-1)=0 THEN C=A-1 :: D=B-1
 :: GOSUB 1160 :: IF NJ=0 THEN RETURN
1300 IF Z(A-1,B+1)=0 THEN C=A-1 :: D=B+1
 :: GOSUB 1160 :: IF NJ=0 THEN RETURN
1310 IF Z(A,B)<>-2 THEN RETURN ELSE IF Z
(A+1,B-1)=0 THEN C=A+1 :: D=B-1 :: GOSUB
 1160 :: IF NJ=0 THEN RETURN
1320 IF Z(A-1,B-1)=0 THEN C=A-1 :: D=B-1
 :: GOSUB 1160 :: RETURN ELSE RETURN
1330 REM CK REQUIRED JUMP UP M
1340 RJ1,RJ2,RJ3,RJ4=0 :: IF A+2>8 THEN
1390 ELSE IF B-2<1 THEN 1370
1350 IF Z(A+1,B-1)=OPP AND Z(A+2,B-2)=0
THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ4=B-
2 :: GOTO 1390
1360 IF Z(A+1,B-1)=OPP+OPP AND Z(A+2,B-2
)=0 THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ
4=B-2 :: GOTO 1390
1370 IF B+2>8 THEN 1390 ELSE IF Z(A+1,B+
1)=OPP AND Z(A+2,B+2)=0 THEN RJ1=A :: RJ
2=B :: RJ3=A+2 :: RJ4=B+2
1380 IF Z(A+1,B+1)=OPP+OPP AND Z(A+2,B+2
)=0 THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ
4=B+2
1390 RETURN
1400 REM CK REQUIRED JUMP UP P
1410 RJ1,RJ2,RJ3,RJ4=0 :: IF A+2>8 THEN
1460 ELSE IF B+2>8 THEN 1440
1420 IF Z(A+1,B+1)=OPP AND Z(A+2,B+2)=0
THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ4=B+
2 :: GOTO 1460
```

```
143Ø IF Z(A+1,B+1)=OPP+OPP AND Z(A+2,B+2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ
4=B+2 :: GOTO 146Ø
144Ø IF B-2<1 THEN 146Ø ELSE IF Z(A+1,B-
1)=OPP AND Z(A+2,B-2)=Ø THEN RJ1=A :: RJ
2=B :: RJ3=A+2 :: RJ4=B-2 :: GOTO 146Ø
145Ø IF Z(A+1,B-1)=OPP+OPP AND Z(A+2,B-2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A+2 :: RJ
4=B-2
146Ø RETURN
147Ø REM CK REQUIRED JUMP DOWN M
148Ø RJ1,RJ2,RJ3,RJ4=Ø :: IF A-2<1 THEN
153Ø ELSE IF B-2<1 THEN 151Ø
149Ø IF Z(A-1,B-1)=OPP AND Z(A-2,B-2)=Ø
THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ4=B-
2 :: GOTO 153Ø
15ØØ IF Z(A-1,B-1)=OPP+OPP AND Z(A-2,B-2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ
4=B-2 :: GOTO 153Ø
151Ø IF B+2>8 THEN 153Ø ELSE IF Z(A-1,B+
1)=OPP AND Z(A-2,B+2)=Ø THEN RJ1=A :: RJ
2=B :: RJ3=A-2 :: RJ4=B+2
152Ø IF Z(A-1,B+1)=OPP+OPP AND Z(A-2,B+2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ
4=B+2
153Ø RETURN
154Ø REM CK REQUIRED JUMP DOWN P
155Ø RJ1,RJ2,RJ3,RJ4=Ø :: IF A-2<1 THEN
16ØØ ELSE IF B+2>8 THEN 158Ø
156Ø IF Z(A-1,B+1)=OPP AND Z(A-2,B+2)=Ø
THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ4=B+
2 :: GOTO 16ØØ
157Ø IF Z(A-1,B+1)=OPP+OPP AND Z(A-2,B+2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ
4=B+2 :: GOTO 16ØØ
158Ø IF B-2<1 THEN 16ØØ ELSE IF Z(A-1,B-
1)=OPP AND Z(A-2,B-2)=Ø THEN RJ1=A :: RJ
2=B :: RJ3=A-2 :: RJ4=B-2 :: GOTO 16ØØ
159Ø IF Z(A-1,B-1)=OPP+OPP AND Z(A-2,B-2
)=Ø THEN RJ1=A :: RJ2=B :: RJ3=A-2 :: RJ
4=B-2
16ØØ RETURN
161Ø REM DISPLAY MY MOVE
162Ø CALL HCHAR(5,3,B+64):: CALL HCHAR(5
,5,A+48):: CALL HCHAR(5,8,D+64):: CALL H
```

```
CHAR(5,10,C+48):: RETURN
1630 REM PLACE ON SCREEN
1640 ROW=16-(X*2)+5 :: COL=Y*2+13 :: IF
Z(X,Y)<>0 THEN 1670 ELSE IF (ROW+3)/4=IN
T((ROW+3)/4)AND(COL+3)/4=INT((COL+3)/4)T
HEN 1660
1650 IF (ROW+1)/4=INT((ROW+1)/4)AND(COL+
1)/4=INT((COL+1)/4)THEN 1660 ELSE 1710
1660 CALL HCHAR(ROW,COL,116,2):: CALL HC
HAR(ROW+1,COL,116,2):: GOTO 1710
1670 IF Z(X,Y)<0 THEN I=120 ELSE I=112
1680 CALL HCHAR(ROW,COL,I):: CALL HCHAR(
ROW+1,COL,I+1):: CALL HCHAR(ROW+1,COL+1,
I+2):: CALL HCHAR(ROW,COL+1,I+3)
1690 IF Z(X,Y)=-2 THEN I=125 ELSE IF Z(X
,Y)=2 THEN I=117 ELSE 1710
1700 CALL HCHAR(ROW+1,COL,I):: CALL HCHA
R(ROW+1,COL+1,I+1)
1710 RETURN
1720 REM GAME OVER
1730 IF TIME<0 THEN TOUT=TOUT+1 :: CALL
SOUND(250,250,1):: TIME=LIM :: GOTO 810
1740 IF B1=14 THEN MSG1$="YOU RESIGNED..
...GAME OVER" :: GOTO 1760
1750 IF YOU<ME THEN MSG1$="I WIN........
....GAME OVER" ELSE MSG1$="YOU WIN......
....GAME OVER"
1760 DISPLAY AT(2,1):MSG1$
1770 IF TOUT>0 THEN DISPLAY AT(1,1):"YOU
 TIMED OUT";TOUT;"TIMES"
1780 CALL KEY(3,KEY,STAT):: DISPLAY AT(2
3,1):" " :: DISPLAY AT(23,1):"   ENTER
TO BEGIN NEW GAME" :: IF STAT=0 THEN 176
0
1790 DISPLAY AT(1,1):" " :: DISPLAY AT(2
,1):" " :: DISPLAY AT(23,1):" " :: CALL
VCHAR(1,1,32,24*12):: GOTO 240
1800 REM HEADER
1810 SUB HEADER
1820 CALL CLEAR :: CALL SCREEN(8):: RAND
OMIZE :: CALL MAGNIFY(4)
1830 CALL CHAR(36,"3C4299A1A199423CF9999
999FF999999")
1840 CALL CHAR(135,"183C7EFFFF7E3C18183C
7EFFFF7E3C18")
```

```
185Ø CALL CHAR(128,"FFFFFFFFFFFFFFFFØØØØ
ØØØØØØØØØØØØØØØØØØØØØØØØØØØØFFFFFFFFFFFF
FFFF")
186Ø CALL CHAR(124,"ØØØØØØØØØØØØØØØØFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFØØØØØØØØØØØØ
ØØØØ")
187Ø DISPLAY AT(8,1Ø):"CHECKERS"
188Ø DISPLAY AT(1Ø,4):"By A. Hawley"
189Ø DISPLAY AT(12,4):"Allan's Single Sy
stem"
19ØØ DISPLAY AT(15,14):"$ % 1983" :: I=1
28 :: FOR A=1 TO 15 :: IF I=124 THEN I=1
28 ELSE I=124
191Ø CALL SPRITE(#A,I,A+1,INT(RND*192)+1
,INT(RND*256)+1,INT(RND*1ØØ)-5Ø,INT(RND*
1ØØ)-5Ø):: NEXT A
192Ø CALL KEY(Ø,K,S):: IF S=Ø THEN 192Ø
193Ø CALL DELSPRITE(ALL):: CALL MAGNIFY(
1):: CALL CHARSET :: SUBEND
```

TANK ABOUT MATH

Tank about math is a creative exercise in simple math for children that teaches multiplication and division. Rewards are built into the lessons making the problems fun to work.

When the first screen appears, there are three gunners on the right side and a tank on the left. The screen displays a random multiplication problem using two numbers between one and ten. When you enter an answer to the problem, one of your gunners fires at the tank. If the answer is lower than the correct one, the shot falls below the tank. If the answer is too high then the shot passes above. With each wrong answer the tank takes a shot hitting one of your cannons untill three wrong guesses are counted. The computer then gives you the right answer and a new problem appears.

After five correct answers the computer allows you to choose whether you wouldlike to do multiplication or division and the upper limit for random numbers increases to fifteen. For each five sucessful answers the numbers used in the lesson increase until they reach 25. A good suggestion for this program would be to add a short game or musical tune as a reward for each set of problems correctly answered.

```
100 REM TANK ABOUT MATH
110 REM ALLEN HOLLEY C 1983
120 REM CRACKING THE 99/4A
130 CALL CLEAR :: CALL SCREEN(8)
140 DISPLAY AT(4,6):"TANK ABOUT MATH"
150 DISPLAY AT(6,1):" By A. Hawley"
160 DISPLAY AT(8,1):" Allan's Single Sys
tem"
170 DISPLAY AT(10,20):"  `1983"
180 CALL CHAR(93,"FFFFFFFFFFFFFFFF")
190 CALL CHAR(95,"F9999999FF999999")
200 CALL CHAR(96,"3C4299A1A199423C")
210 FOR I=24 TO 1 STEP -1 :: IF I=6 THEN
 230
```

```
220 FOR II=32 TO 1 STEP -1 :: CALL HCHAR
(I,II,93):: NEXT II
230 NEXT I :: FOR II=32 TO 1 STEP -1 ::
CALL HCHAR(6,II,93):: NEXT II
240 CALL CHAR(128,"FC00FCFFFFFC00FC")
250 CALL CHAR(129,"0000001818")
260 CALL CHAR(136,"0183C7EF7F3F7FFF")
270 CALL CHAR(137,"00001038 7C3810")
280 CALL CHAR(125,"894A2CF81F345291")
290 CALL CHAR(124,"8142241818244281")
300 CALL CHAR(94,"0000100007C0010")
310 CALL CLEAR :: CALL COLOR(13,9,8)::CA
LL COLOR(14,3,8)
320 REM MAIN PROGRAM
330 CALL MAGNIFY(2):: MD,MS=1 ::LEVEL=15
 :: PLAYLEVEL=1 :: ON WARNING NEXT
340 IF COUNT<5 THEN 440 ELSE COUNT=0 ::
IF LEVEL>5 THEN LEVEL=LEVEL-5 ELSE LEVEL
=LEVEL-1
350 REM CHANGE LEVELS
360 PLAYLEVEL=PLAYLEVEL+1 :: CALL DELSPR
ITE(ALL):: CALL CLEAR :: CALL SCREEN(9)
370 DISPLAY AT(8,6):"CONGRATULATIONS!!!"
 :: DISPLAY AT(10,10):"5 IN A ROW"
380 DISPLAY AT(14,7):"WELCOME TO LEVEL"
 :: DISPLAY AT(16,13):PLAYLEVEL
390 CALL HCHAR(1,1,42,96):: CALL VCHAR(1
,30,42,72):: CALL HCHAR(22,1,42,96):: CA
LL VCHAR(1,1,42,72)
400 CALL SOUND(300,392,1):: CALL SOUND(3
00,349,1):: CALL SOUND(300,333,1):: CALL
 SOUND(500,392,1):: CALL SOUND(300,196,1
)
410 CALL SOUND(700,220,1):: CALL SOUND(1
000,300,30):: CALL SOUND(1,300,30):: CAL
L SCREEN(8):: CALL CLEAR
420 DISPLAY AT(10,6):"1  MULTIPLICATION"
 :: DISPLAY AT(12,6)BEEP:"2  DIVISION"
430 CALL KEY(3,MD,STAT):: IF STAT=0 THEN
 430 ELSE IF MD<49 OR MD>50 THEN 430 ELS
E MD=MD-48
440 REM SET UP SCREEN
450 CALL CLEAR :: CALL SPRITE(#10,128,9,
12*8,1*8)
460 CALL SPRITE(#1,136,3,6*8,29*8)::CALL
```

```
 SPRITE(#2,136,3,12*8,29*8):: CALL SPRIT
E(#3,136,3,18*8,29*8)
470 CALL CLEAR :: DISPLAY AT(22,1):"Pres
s key to enter distance"
480 RANDOMIZE :: CALL PEEK(-31880,N1)::
N1=INT(N1/LEVEL+1):: RANDOMIZE :: CALL P
EEK(-31880,N2):: N2=INT(N2/LEVEL)::N3=N1
*N2
490 REM PLAY
500 ON MD GOTO 520,510
510 DISPLAY AT(2,1):"DISTANCE TO TANK ="
;N3;"^";N2 :: HITGUN=0 :: GOTO 530
520 DISPLAY AT(2,1):"DISTANCE TO TANK ="
;N1;"X";N2 :: HITGUN=0
530 DISPLAY AT(4,1):" "::HITGUN=HITGUN+1
540 GOSUB 590 :: DISPLAY AT(4,10):"ENTER
HERE" :: ACCEPT AT(4,21)VALIDATE(DIGIT)B
EEP:ANS
550 ON MD GOTO 570,560
560 IF ANS=N1 THEN GOSUB 770 :: GOTO 340
 :: ELSE COUNT=0 :: IF ANS<N1 THEN OT=1
:: GOTO 580 :: ELSE OT=2 :: GOTO 580
570 IF ANS=N3 THEN GOSUB 770 :: GOTO 340
 :: ELSE COUNT=0 :: IF ANS<N3 THEN OT=1
ELSE OT=2
580 GOSUB 690 :: GOSUB 640 ::IF HITGUN=3
 THEN 340 ELSE 530
590 REM FIRE TANK (MISS)
600 RANDOMIZE :: CALL PEEK(-31880,C1)::
C1=INT(C1/4+2)::IF C1<9 OR C1>14 THEN 61
0 ELSE 600
610 CALL SOUND(100,220,2):: MS=MS*-1 ::
CALL SPRITE(#11,129,9,12*8,3*8,C1*MS,50)
620 CALL KEY(0,KEY,STAT):: IF STAT<>0 TH
EN 630 :: CALL POSITION(#11,PR11,PC11)::
 IF PC11<8*29 THEN 620 ELSE CALL DELSPRI
TE(#11):: GOTO 600
630 CALL DELSPRITE(#11):: RETURN
640 REM FIRE TANK (HIT)
650 IF HITGUN=2 THEN HG=0 ELSE IF HITGUN
=1 THEN HG=-11 ELSE HG=11
660 CALL SOUND(100,220,2):: CALL SPRITE(
#11,129,9,12*8,3*8,HG,50)
670 CALL SOUND(950,300,30):: CALL SOUND(
1,300,30):: CALL PATTERN(#HITGUN,124)::
```

```
CALL DELSPRITE(#11)
680 CALL SOUND(600,-7,0):: CALL DELSPRIT
E(#HITGUN):: RETURN
690 REM FIRE CANNON (MISS)
700 ON HITGUN GOTO 710,720,730
710 AH3=6 :: IF OT=1 THEN OC=20 :: GOTO
740 :: ELSE OC=15 :: GOTO 740
720 AH3=12 :: IF OT=1 THEN OC=15 :: GOTO
 740 :: ELSE OC=12 :: GOTO 740
730 AH3=18 :: IF OT=1 THEN OC=11 ELSE OC
=9
740 CALL SOUND(300,-3,2):: CALL SPRITE(#
4,129,13,AH3*8-8,28*8-4,-10,-10)
750 FOR AH=1 TO OC :: CALL SOUND(10,300,
30):: CALL SOUND(1,300,30):: CALL MOTION
(#4,-10+AH,-10):: NEXT AH
760 CALL POSITION(#4,PR11,PC11)::IF PC11
>4 THEN 760 ELSE CALL DELSPRITE(#4):: RE
TURN
770 REM FIRE CANNON (HIT)
780 ON HITGUN GOTO 790,800,810
790 COUNT=COUNT+1 :: AH3=6 :: OC=17 :: G
OTO 820
800 AH3=12 :: OC=13 :: GOTO 820
810 AH3=18 :: OC=10
820 CALL SOUND(300,-3,2):: CALL SPRITE(#
4,129,13,AH3*8-8,28*8-4,-10,-10)
830 FOR AH=1 TO OC :: CALL SOUND(10,300,
30):: CALL SOUND(1,300,30):: CALL MOTION
(#4,-10+AH,-10):: NEXT AH
840 REM CALL POSITION(#4,PR11,PC11):: IF
 PC11>4 THEN 2980 ELSE CALL DELSPRITE(#4
):: RETURN
850 CALL PEEK(-31877,AH9)::IF AH9 AND 32
 THEN 860 ELSE 850
860 CALL PATTERN(#10,125):: CALL DELSPRI
TE(#4):: CALL SOUND(600,-7,0):: CALL SOU
ND(1,300,30):: CALL DELSPRITE(#10):: RET
URN
```

Extended Basic
Speech unit optional

HANGMAN FOR TWO

Hangman is a computer version of the game we all played as children where one player would draw a gallows and some dashes representing the letters of a word, and the other player would try to guess the word by guessing the letters. If the letter guessed was not among those that made up the word, a part of a hanging man was added to the picture. The object was to guess the word before the hanging man was completely drawn. Sometimes, the hanging man had a head, body, arms, legs, and all his fingers and toes. This, of course, gave the guesser a chance to win. In this version of hangman, you only have seven guesses before you are completely hung. This computer version has several features not found in other computer versions:

1. Automatic scorekeeping
2. The ability to play against time
3. Entry correction
4. Automatic round keeping
5. The ability to enter phrases
6. Automatic blank removal

Automatic score keeping

The alphabet is displayed on the top of the screen with a number under each letter. This number is the amount added to the guesser's score each time that letter is successfully guessed. If a letter is guessed and the word contains two of that letter, twice that letter's value is added to the guesser's score.

If the entire word or phrase is guessed, the guesser receives four points for every unguessed letter remaining in the word or phrase. Therefore it is advantageous for a guesser to guess the entire word or phrase when he can. The penalty, if wrong, is the same as missing a guess for one letter. If a guesser enters more than one letter as a guess, HANGMAN assumes the guesser is attempting to guess the entire word or phrase and compares what was entered against the word or phrase to be guessed.

Playing against time.

When the game begins, HANGMAN asks for the players' names and then HANGMAN asks agenst what time interval they

wish to play. Values from Ø to 999 are accepted, but values between 1Ø and 3Ø seconds are recommended. A countdown of the seconds are shown on the left-hand side of the gallows picture.

The guesser has only the amount of time specified to make a guess; if he does not guess correctly, another piece is added to the hanging man. When playing against time the LEFT ARROW and the ERASE keys are operational in addition to entering upper-case letters and spaces. This was done to ensure easy correction of entries.

Entry correction.

If a player enters a word or phrase to be guessed and realizes a mistake has been made, or changes his mind, the entry can be corrected by entering 'CE', for "correct entry." This can be done only if no guess has yet been made.

Automatic rounds.

This game is played in rounds. One round is complete when each player has guessed one time. Any round number ending in .5 indicates the middle of a round.

Enter phrases.

In most versions of HANGMAN, blanks (spaces) are not allowed. Consequently only single words could be entered. This version allows blanks; Thus you can enter phrases such as "OVER THE RAINBOW." For example, you may wish to play a game in which only the names of movies are allowed. Up to 24 upper-case letters and blanks can be entered.

Automatic blank removal

If a player accidentally enters more than one blank between words when entering a phrase or in guessing a phrase, HANGMAN will automatically remove all but one blank. This helps ensure valid comparisons when a player attempts to guess an entire phrase.

```
100 REM HANGMAN FOR TWO
110 REM COPYRIGHT JOHN COPE 83
120 REM CRACKING THE 99/4A
130 REM SPEACH (OPTIONAL)
140 CALL CLEAR
150 CALL SCREEN(12)
160 A1$="ABCDEFGHIJKLM" :: A2$="NOPQRSTU
VWXYZ"
170 DISPLAY AT(1,3):A1$&A2$
180 N1$="1332142418513" :: N2$="11391111
44849"
190 DISPLAY AT(2,3):N1$&N2$
```

```
200 GOSUB 2350 !TIMING ?
210 DISPLAY AT(5,2):"NAME 1 (1 TO 6 LETT
ERS)"
220 ACCEPT AT(6,2)SIZE(6):NAME1$
230 L=LEN(NAME1$):: L=INT((6-L)/2):: CN1
=3+L
240 DISPLAY AT(6,2)SIZE(8)
250 DISPLAY AT(6,CN1):NAME1$
260 DISPLAY AT(5,2):"NAME 2  (1 TO 6 LET
TERS)"
270 ACCEPT AT(6,11)SIZE(6):NAME2$
280 L=LEN(NAME2$):: L=INT((6-L)/2):: CN1
=12+L
290 DISPLAY AT(6,11)SIZE(8)
300 DISPLAY AT(6,CN1):NAME2$
310 DISPLAY AT(5,2)SIZE(24)
320 DISPLAY AT(8,4):USING "###":000
330 DISPLAY AT(8,13):USING "###":000
340 DISPLAY AT(11,3):"LETTERS GUESSED"
350 CALL COLOR(13,6,6):: CALL COLOR(14,3
,3)
360 R=4 :: FOR I=1 TO 8
370 CALL HCHAR(R,21,128,10)
380 R=R+1 :: NEXT I
390 R=12 :: FOR I=1 TO 7
400 CALL HCHAR(R,21,136,10)
410 R=R+1 :: NEXT I
420 CALL COLOR(12,2,6):: CALL COLOR(11,2
,3)
430 CALL CHAR(120,"3C3C3C3C3C3C3C3C")
440 CALL CHAR(121,"FFFFFFFF00000000")
450 CALL CHAR(122,"FFFFFFFFFFFFFFFF")
460 CALL CHAR(123,"F0F0F0F0F0F0F0F0")
470 CALL CHAR(124,"0F0F0F0F0F0F0F0F")
480 CALL CHAR(125,"0000FFFFFFFF0000")
490 CALL CHAR(126,"3C3CFFFFFFFF3C3C")
500 CALL CHAR(112,"3C3C3C3C3C3C3C3C")
510 CALL CHAR(113,"FFFFFFFFFFFFFFFF")
520 CALL CHAR(114,"FFFF0000000000000")
530 CALL CHAR(115,"FCFC3C3C3C3C3C3C")
540 CALL CHAR(116,"3F3F3C3C3C3C3C3C")
550 CALL CHAR(127,"FFFFFFFF3C3C3C3C")
560 CALL CHAR(95,"7E7E7E7E7E7E7E7E")
570 CALL HCHAR(4,24,127)
580 CALL HCHAR(4,25,121,3)
```

```
590 CALL HCHAR(4,28,127)
600 CALL VCHAR(5,28,120,7)
610 CALL VCHAR(12,28,112,6)
620 CALL HCHAR(18,26,113,5)
630 DISPLAY AT(19,3)SIZE(1):"*"
640 OPTION BASE 1
650 DIM LG$(26),L$(24),TLG$(24)
660 A1R=22 :: A1C=1 :: A2R=23 :: A2C=1
670 ROUND=.5 :: L=0
680 WCOLS=2 :: WROWS=20 :: SC1R=8 :: SC1
C=4 :: SC2R=8 :: SC2C=13
690 RANDOMIZE :: N=10-(10*RND):: FOR I=1
 TO N :: P=INT(1+(2*RND)):: NEXT I
700 REM CHANGE TURNS
710 ROUND=ROUND+.5 :: NHITS=0 :: NNB=0 :
: DRAW=0
720 DISPLAY AT(4,7)SIZE(12):"ROUND";ROUN
D
730 DISPLAY AT(13,3)SIZE(15):: DISPLAY A
T(15,3)SIZE(15):: DISPLAY AT(17,3)SIZE(1
5):: DISPLAY AT(20,3)SIZE(24)
740 CALL VCHAR(5,24,128,7):: CALL VCHAR(
7,23,128,2):: CALL VCHAR(7,25,128,2):: C
ALL HCHAR(10,22,128,5)
750 CALL VCHAR(12,24,136):: CALL HCHAR(1
3,23,136,3):: CALL VCHAR(14,23,136,3)::
CALL VCHAR(14,25,136,3)
760 FOR I=1 TO 26 :: LG$(I)="NULL" :: NE
XT I
770 IF P=1 THEN P=2 ELSE P=1
780 IF P=1 THEN GIVER$=NAME1$ ELSE GIVER
$=NAME2$
790 IF GIVER$=NAME1$ THEN GUESSER$=NAME2
$ ELSE GUESSER$=NAME1$
800 DISPLAY AT(A1R,A1C):">> "&GUESSER$&"
, TURN YOUR HEAD"
810 DISPLAY AT(A2R,A2C):">> "&GIVER$&",
ENTER WORD/PHRASE"
820 ACCEPT AT(20,3)SIZE(24)VALIDATE(UALP
HA," "):PHRASE$
830 PARM=1 :: GOSUB 2180
840 IF LEN(PHRASE$)=0 THEN 800
850 IF ASC(PHRASE$)=32 THEN 800
860 DISPLAY AT(20,3)SIZE(24)
870 L=LEN(PHRASE$)
```

```
880 FOR I=1 TO L
890 L$(I)=SEG$(PHRASE$,I,1)
900 IF ASC(L$(I))<>32 THEN NNB=NNB+1
910 NEXT I
920 R=20 :: C=3 :: FOR I=1 TO L :: IF L$
(I)=" " THEN 930 ELSE DISPLAY AT(R,C):"_
"
930 C=C+1 :: NEXT I
940 LGROW=13 :: LGCOL=3
950 G=0
960 REM GET A NEW LETTER
970 G=G+1
980 GOSUB 2790 !CLEAR AREA
990 IF T$="YES" THEN GOSUB 2450 :: GOTO
1020
1000 DISPLAY AT(A1R,A1C):">> "&GUESSER$&
",ENTER LETTER/GUESS"
1010 ACCEPT AT(A2R,A2C)SIZE(24)VALIDATE(
UALPHA," "):LG$(G)
1020 IF G>1 THEN 1030 :: IF LG$(G)<>"CE"
 THEN 1030 :: NNB=0 :: GOTO 800
1030 IF LEN(LG$(G))=0 THEN 980
1040 F$=SEG$(LG$(G),1,1):: IF F$=" " THE
N 980
1050 IF LEN(LG$(G))=1 THEN 1060 ELSE GPH
RASE$=LG$(G):: G=G-1 :: PARM=2 :: GOSUB
2180 :: GOTO 1670
1060 FOR I=1 TO G-1
1070 IF LG$(G)<>LG$(I)THEN 1120 ELSE CAL
L SOUND(500,440,4)
1080 GOSUB 2790
1090 DISPLAY AT(A1R,A1C)SIZE(28):">> ERR
OR. LETTER GUESSED"
1100 FOR I=1 TO 350 :: NEXT I
1110 G=G-1 :: VRC=1 :: GOTO 960
1120 NEXT I :: VRC=0
1130 DISPLAY AT(LGROW,LGCOL)SIZE(1):LG$(
G)
1140 LGCOL=LGCOL+2
1150 IF LGCOL<18 THEN 1170
1160 LGCOL=3 :: LGROW=LGROW+2
1170 GOTO 1180
1180 REM CHECK IF HIT
1190 RC=0
1200 FOR J=1 TO L
```

```
1210 IF LG$(G)=L$(J)THEN GOSUB 1250
1220 NEXT J
1230 REM CHECK IF MATCH
1240 IF RC<>0 THEN GOSUB 1280 ELSE GOSUB
 1760 :: GOTO 960 ! GET ANOTHER LETTER
1250 REM MATCH: SHOW LETTERS
1260 RC=RC+1 :: WCOL=WCOLS+J :: NHITS=NH
ITS+1
1270 DISPLAY AT(WROWS,WCOL)SIZE(1):LG$(I
):: RETURN
1280 REM MATCH: FIND SCORE
1290 CALL SAY("GOOD GUESS")
1300 B=ASC(LG$(I))-64
1310 IF B=26 THEN 1460
1320 ON B GOTO 1330,1340,1350,1360,1370,
1380,1390,1400,1410,1420,1430,1440,1450,
1470,1480,1490,1500,1510,1520,1530,1540,
1550,1560,1570,1580
1330 S=(1*RC):: GOTO 1610
1340 S=(3*RC):: GOTO 1610
1350 S=(3*RC):: GOTO 1610
1360 S=(2*RC):: GOTO 1610
1370 S=(1*RC):: GOTO 1610
1380 S=(4*RC):: GOTO 1610
1390 S=(2*RC):: GOTO 1610
1400 S=(4*RC):: GOTO 1610
1410 S=(1*RC):: GOTO 1610
1420 S=(8*RC):: GOTO 1610
1430 S=(5*RC):: GOTO 1610
1440 S=(1*RC):: GOTO 1610
1450 S=(3*RC):: GOTO 1610
1460 S=(9*RC):: GOTO 1610
1470 S=(1*RC):: GOTO 1610
1480 S=(1*RC):: GOTO 1610
1490 S=(3*RC):: GOTO 1610
1500 S=(9*RC):: GOTO 1610
1510 S=(1*RC):: GOTO 1610
1520 S=(1*RC):: GOTO 1610
1530 S=(1*RC):: GOTO 1610
1540 S=(1*RC):: GOTO 1610
1550 S=(4*RC):: GOTO 1610
1560 S=(4*RC):: GOTO 1610
1570 S=(8*RC):: GOTO 1610
1580 S=(4*RC):: GOTO 1610
1590 S=(9*RC):: GOTO 1610
```

```
1600 S=(9*RC)
1610 IF P=1 THEN SCORE2=SCORE2+S ELSE SC
ORE1=SCORE1+S
1620 GOSUB 2150
1630 IF NHITS<NNB THEN RETURN ELSE GOSUB
 2790
1640 DISPLAY AT(A1R,A1C):">> "&GUESSER$&
", YOU GOT THEM ALL"
1650 CALL SAY("YOU GOT M ALL"):: DISPLAY
 AT(A2R,A2C):"   PRESS ANY KEY"
1660 CALL KEY(0,RV,SV):: IF SV=0 THEN 16
60 ELSE GOTO 700
1670 IF GPHRASE$=PHRASE$ THEN DISPLAY AT
(WROWS,WCOLS+1):GPHRASE$ ELSE GOTO 1700
1680 IF GPHRASE$=PHRASE$ THEN GOSUB 2790
 :: DISPLAY AT(A1R,A1C):">> CORRECT !!"
:: DISPLAY AT(A2R,A2C):"   PRESS ANY KEY
"
1690 CALL SAY("CORRECT GOOD WORK"):: GOT
O 1730
1700 GOSUB 1760 :: CALL SAY("SORRY NOT C
ORRECT"):: DISPLAY AT(A1R,A1C):">> SORRY
, WRONG"
1710 CALL SOUND(500,440,5):: FOR I=1 TO
350
1720 NEXT I :: GOTO 960
1730 IF P=1 THEN SCORE2=SCORE2+(4*(NNB-N
HITS))ELSE SCORE1=SCORE1+(4*(NNB-NHITS))
1740 GOSUB 2150
1750 CALL KEY(0,RV,SV):: IF SV=0 THEN 17
50 :: GOTO 700
1760 REM NO MATCH
1770 CALL SAY("UHOH")
1780 DRAW=DRAW+1
1790 ON DRAW GOSUB 1860,1920,1980,2010,2
040,2090,2120
1800 IF DRAW<7 THEN RETURN
1810 DISPLAY AT(WROWS,WCOLS+1)SIZE(24):P
HRASE$
1820 GOSUB 2790
1830 DISPLAY AT(A1R,A1C):">> "&GUESSER$&
", YOU'RE HUNG !!" :: CALL SAY("SO SORRY
")
1840 DISPLAY AT(A2R,A2C):"   PRESS ANY K
EY"
```

```
1850 CALL KEY(0,RV,SV):: IF SV=0 THEN 18
50 :: GOTO 700
1860 REM DRAW HEAD
1870 CALL VCHAR(7,23,124,2)
1880 CALL VCHAR(7,24,122,2)
1890 CALL VCHAR(7,25,123,2)
1900 CALL VCHAR(5,24,120,2)
1910 RETURN
1920 REM DRAW BODY
1930 CALL HCHAR(9,24,120)
1940 CALL HCHAR(10,24,126)
1950 CALL HCHAR(12,24,112)
1960 CALL HCHAR(11,24,120)
1970 RETURN
1980 REM DRAW RIGHT ARM
1990 CALL HCHAR(10,22,125,2)
2000 RETURN
2010 REM DRAW LEFT ARM
2020 CALL HCHAR(10,25,125,2)
2030 RETURN
2040 REM DRAW HIPS
2050 CALL HCHAR(13,23,116)
2060 CALL HCHAR(13,24,114)
2070 CALL HCHAR(13,25,115)
2080 RETURN
2090 REM DRAW RIGHT LEG
2100 CALL VCHAR(14,23,112,3)
2110 RETURN
2120 REM DRAW LEFT LEG
2130 CALL VCHAR(14,25,112,3)
2140 RETURN
2150 DISPLAY AT(SC1R,SC1C)SIZE(3):USING
"###":SCORE1
2160 DISPLAY AT(SC2R,SC2C)SIZE(3):USING
"###":SCORE2
2170 RETURN
2180 REM REMOVE EXCESS                  SPA
CES
2190 ON PARM GOTO 2200,2220
2200 NPHRASE$=PHRASE$
2210 GOTO 2230
2220 NPHRASE$=GPHRASE$
2230 P1=POS(NPHRASE$," ",1)
2240 P2=POS(NPHRASE$," ",P1+1):: DIF=P2-
P1
```

```
2250 IF P1=0 OR P2=0 THEN 2280
2260 IF DIF=1 THEN 2290
2270 P1=P2 :: GOTO 2240
2280 ON PARM GOTO 2310,2330
2290 NPHRASE$=SEG$(NPHRASE$,1,P1)&SEG$(N
PHRASE$,P2+1,LEN(NPHRASE$))
2300 GOTO 2230
2310 PHRASE$=NPHRASE$
2320 GOTO 2340
2330 GPHRASE$=NPHRASE$
2340 RETURN
2350 REM ASK IF TIMING IS WANTED
2360 DISPLAY AT(5,2)SIZE(28):"TIME LIMIT
 ON GUESSING? Y/N"
2370 CALL KEY(0,RV,SV)
2380 IF SV=0 THEN 2370
2390 IF RV=89 THEN T$="YES" :: GOTO 2420
2400 IF RV=78 THEN T$="NO" :: RETURN
2410 GOTO 2360
2420 DISPLAY AT(5,2)SIZE(28):"ENTER INTE
RVAL IN SECONDS"
2430 ACCEPT AT(6,2)SIZE(3)VALIDATE(DIGIT
):INTVL
2440 RETURN
2450 REM ACCEPT LETTER OR GUESS ON TIMER
2460 IF VRC=1 THEN INTER=INTER-1 :: T=1
:: GOTO 2480
2470 INTER=INTVL :: GOSUB 2790 :: T=1 ::
 IF LA=1 THEN E=E :: GOTO 2480 ELSE E,LA
=0
2480 DISPLAY AT(A1R,A1C):">> "&GUESSER$&
",ENTER LETTER/GUESS"
2490 FOR I=1 TO INTER
2500 DISPLAY AT(18,20)SIZE(4):INTER
2510 FOR J=1 TO 27
2520 CALL KEY(0,RV,SV)
2530 IF SV=-1 THEN GOTO 2560
2540 NEXT J
2550 INTER=INTER-1 :: NEXT I :: CALL SAY
("OUT OF TIME"):: GOSUB 1760 :: GOTO 247
0
2560 IF RV=32 AND T=1 THEN 2540 :: IF RV
=32 AND T<>1 THEN 2570 :: IF RV<65 THEN
2620 :: IF RV>90 THEN 2620 :: LA=0
2570 TLG$(T)=CHR$(RV):: DISPLAY AT(A2R,A
```

```
2C+T+1):TLG$(T)
2580 DISPLAY AT(A2R,A2C+T+2):"_" :: T=T+
1
2590 IF T>24 THEN CALL SOUND(500,440,5):
: T=T-1 :: GOTO 2480 ELSE GOTO 2540
2600 FOR I=1 TO 350 :: NEXT I :: GOTO 24
70
2610 GOTO 2540
2620 REM NON-ALPHA INPUT
2630 IF RV=13 THEN GOTO 2700
2640 IF RV=8 THEN T=T-2 :: IF T<0 THEN T
=0
2650 IF RV=8 THEN 2580
2660 IF RV=7 THEN 2770 ELSE T=T-1 :: IF
T<=0 THEN T=1
2670 CALL SOUND(500,440,5)
2680 REM DISPLAY AT(A1R,A1C):">> INVALID
 CHARACTER" :: FOR I=1 TO 350 :: NEXT I
2690 DISPLAY AT(A1R,A1C):GUESSER$&",ENTE
R LETTER/GUESS" :: INTER=INTER-1 :: GOTO
 2490
2700 REM CHECK FOR HIT. IF 1 {ENTER  CHA
RACTER ENTERED,  PRESSED}THEN ASSUME HE'
S             GUESSING A LETTER
2710 IF T=1 THEN GOSUB 2790 :: DISPLAY A
T(A1R,A1C):GUESSER$&",ENTER LETTER/GUESS
" :: INTER=INTER-1 :: GOTO 2490
2720 IF T=2 THEN LG$(G)=TLG$(T-1):: RETU
RN
2730 TGPH1$=TLG$(1)&TLG$(2)&TLG$(3)&TLG$
(4)&TLG$(5)&TLG$(6)&TLG$(7)&TLG$(8)&TLG$
(9)&TLG$(10)&TLG$(11)&TLG$(12)
2740 TGPH2$=TLG$(13)&TLG$(14)&TLG$(15)&T
LG$(16)&TLG$(17)&TLG$(18)&TLG$(19)&TLG$(
20)&TLG$(21)&TLG$(22)&TLG$(23)
2750 TGPH3$=TLG$(24)
2760 LG$(G)=SEG$(TGPH1$&TGPH2$&TGPH3$,1,
T-1):: RETURN
2770 REM ERASE
2780 LA=1 :: E=E+1 :: IF E<=2 THEN GOTO
2470 ELSE T=1 :: GOSUB 2790 :: GOTO 2480
2790 REM CLEAR AREAS
2800 DISPLAY AT(A1R,A1C):: DISPLAY AT(A2
R,A2C)
2810 RETURN
```

OTHELLO

Brian Prothro

Othello is a popular game in which a player attempts to end up with the majority of pieces on the board. A move consists of "outflanking" your opponent, flipping his disk(s) over to your color. To outflank means to place a move so that your opponent's row (or rows) of disk(s) is boardered at each end by a disk of your color.

EXAMPLE: A X X X B

Where; "X" stands for your opponents pieces.
"A" is your piece already on the board.
"B" is your move to outflank your opponents row.

All pieces then flip over and become your colored pieces.

In this version of Othello you play the computer, but don't be deceived. The computer does some thinking before it moves. A move is made by entering the desired row and column you want to appear on. A capture must be made on each turn or your move must be forfeited. A disk may outflank any number of disks in one or more rows, and includes the diagional directions as well. Be careful, unexpected results may change a great strategy when the computer makes a move that takes two or even three rows of your pieces!

```
100 REM OTHELLO
110 REM CRACKING THE 99/4A
120 DIM A(9,9),R4(8),C4(8),C$(8),D$(2)
130 CALL CLEAR
140 CALL COLOR(8,9,16,13,16,15,14,5,16)
150 CALL SCREEN(6)
160 GOSUB 2580
170 GOSUB 2830
180 PRINT "Display instructions? (y/n)"
190 CALL KEY(0,KEY,ST):: IF ST=0 THEN 19
0
200 IF KEY=78 THEN 240
```

```
210 PRINT "you are blue and must        o
utflank the computer.  thismeans you mus
t place your   move so that you"
220 PRINT "sandwich the computers pieceb
etween two of your own.    watch out, on
e move can take up to three rows"
230 REM INITIALIZE
240 F2=0
250 PRINT :"Shall i use my":"best strate
gy? (y/n)?"
260 S2=0
270 CALL KEY(0,KEY,ST):: IF ST=0 THEN 27
0
280 IF KEY=78 THEN 300
290 S2=2
300 PRINT :,:"Wait, while i choose":"a s
trategy..."
310 B=-1
320 W=1
330 D$(B+1)="X"
340 D$(0+1)="."
350 D$(W+1)="O"
360 FOR K=1 TO 8
370 READ R4(K),C4(K),C$(K)
380 NEXT K
390 DATA 0,1,A,-1,1,B,-1,0,C,-1,-1,D,0,-
1,E,1,-1,F,1,0,G,1,1,H
400 A(5,5),A(4,4)=W
410 A(5,4),A(4,5)=B
420 P1,H1=2
430 N1=4
440 Z=0
450 REM PLAYER CHOICES
460 P=W :: H=B
470 PRINT :"Do you want to go first? (y/
n)"
480 CALL KEY(0,KEY,ST):: IF ST=0 THEN 48
0
490 IF KEY=78 THEN 550
500 CALL CLEAR
510 GOSUB 2670
520 REM PRINT INITIAL BOARD
530 GOSUB 2430
540 GOTO 1140
550 CALL CLEAR
```

```
560 GOSUB 2670
570 GOSUB 2430
580 REM COMPUTERS MOVE
590 B1=-1
600 R3,C3=0
610 T1=P
620 T2=H
630 REM LOOK FOR EMPTY SQUARE
640 FOR R=1 TO 8
650 GOSUB 2580
660 FOR C=1 TO 8
670 IF A(R,C)<>0 THEN 890
680 REM DOES SQUARE HAVE PLAYER AS A NEI
GHBOR?
690 GOSUB 2030
700 DISPLAY AT(21,6):"my move...":"  che
cking row ";R
710 IF F1=0 THEN 890
720 REM FOUND PLAYER, HOW MANY CAN WE FL
IP
730 U=-1
740 GOSUB 2150
750 REM EXTRA POINT FOR BOARDER LOCATION
760 IF S1=0 THEN 890
770 IF (R-1)*(R-8)<>0 THEN 790
780 S1=S1+S2
790 IF (C-1)*(C-8)<>0 THEN 820
800 S1=S1+S2
810 REM IS POSITION BETTER THAN BEST SO
FAR?
820 IF S1<B1 THEN 890
830 IF S1>B1 THEN 850
840 REM YES
850 B1=S1
860 R3=R
870 C3=C
880 REM END OF LOOP
890 NEXT C :: NEXT R
900 REM COULD WE DO ANYTHING?
910 IF B1>0 THEN 970
920 REM NO
930 IF Z=1 THEN 1720
940 Z=1
950 GOTO 1140
960 REM MAKE THE MOVE
```

```
970 Z=0
980 DISPLAY AT(21,1):"i move to"
990 DISPLAY AT(21,12):R3;",";C$(C3):""
1000 R=R3
1010 C=C3
1020 U=1
1030 GOSUB 2150
1040 P1=P1+S1+1
1050 H1=H1-S1
1060 N1=N1+1
1070 FOR T=1 TO 420 :: NEXT T
1080 DISPLAY AT(21,1):"i get";S1:"of you
r pieces"
1090 REM PRINT OUT BOARD
1100 GOSUB 2430
1110 REM TEST FOR END OF GAME
1120 IF H1=0 OR N1=64 THEN 1720
1130 REM HUMANS MOVE
1140 T1=H :: T2=P
1150 GOSUB 2580
1160 DISPLAY AT(21,1):"9 to forfit":"0 t
o quit"
1170 DISPLAY AT(21,19):"your move:"
1180 DISPLAY AT(22,20):"row col"
1190 ACCEPT AT(23,21)SIZE(1)BEEP:Q$
1200 IF Q$="0" THEN 2020
1210 IF Q$="9" THEN 1260
1220 IF Q$<"1" OR Q$>"8" THEN 1190
1230 R=VAL(Q$)
1240 ACCEPT AT(23,25)VALIDATE("ABCDEFGH"
)SIZE(1)BEEP:X$
1250 GOTO 1350
1260 GOSUB 2580
1270 DISPLAY AT(21,1):"you forfit your t
urn? (y/n)"
1280 CALL KEY(0,KEY,ST):: IF ST=0 THEN 1
280
1290 IF KEY<>89 THEN 1150
1300 GOSUB 2580
1310 DISPLAY AT(21,1):"great, i'll go."
1320 IF Z=1 THEN 1720
1330 Z=1
1340 GOTO 590
1350 FOR C=1 TO 8
1360 IF C$(C)=X$ THEN 1400
```

```
1370 NEXT C
1380 GOTO 1190
1390 REM CHECK IF BLANK
1400 IF A(R,C)=0 THEN 1450
1410 GOSUB 2580
1420 DISPLAY AT(21,1):"That's occupied,
try again."
1430 GOTO 1190
1440 REM CHECK FOR LEGAL NEIGHBOR
1450 GOSUB 2030
1460 IF F1=1 THEN 1510
1470 GOSUB 2580
1480 DISPLAY AT(21,1):"That's not next t
o one of mypieces, try again."
1490 GOTO 1190
1500 REM CHECK IF LEGAL RUN
1510 U=-1
1520 GOSUB 2150
1530 IF S1>0 THEN 1580
1540 GOSUB 2580
1550 DISPLAY AT(21,1):"Sorry, that dosen
't flank   a row, try again."
1560 GOTO 1190
1570 REM IF LEGAL, MAKE PLAYERS MOVE
1580 Z=0
1590 GOSUB 2580
1600 DISPLAY AT(21,1):"That gives you";S
1:"of my pieces"
1610 U=1
1620 GOSUB 2150
1630 H1=H1+S1+1
1640 P1=P1-S1
1650 N1=N1+1
1660 REM PRINT OUT BOARD
1670 GOSUB 2430
1680 REM TEST FOR END OF GAME
1690 IF P1=0 OR N1=64 THEN 1720
1700 GOTO 590
1710 REM END OF GAME DETAILS
1720 GOSUB 2580
1730 DISPLAY AT(20,1):"you have";H1;"pie
ces and":"i have";P1;"pieces"
1740 IF H1=P1 THEN 1780
1750 IF H1>P1 THEN 1800
1760 DISPLAY AT(22,1):"sorry, i won that
```

```
 one"
1770 GOTO 1810
1780 DISPLAY AT(22,1):"a tie"
1790 GOTO 2000
1800 DISPLAY AT(22,1):"i guess you win"
1810 P1=P1-H1
1820 IF P1>0 THEN 1840
1830 P1=-C1
1840 P1=(64*P1)/N1
1850 FOR T=1 TO 300 :: NEXT T
1860 IF P1=0 THEN 1910
1870 IF P1<11 THEN 1930
1880 IF P1<25 THEN 1950
1890 IF P1<39 THEN 1970
1900 IF P1<53 THEN 1990
1910 DISPLAY AT(23,1):"you played a perf
ect game!!!"
1920 GOTO 2000
1930 DISPLAY AT(23,1):"you're great. i t
hought you had never played before!!!"
1940 GOTO 2000
1950 DISPLAY AT(23,1):"pretty good game.
           but i got close!"
1960 GOTO 2000
1970 DISPLAY AT(23,1):"maybe next time.
 i take    the game this time."
1980 GOTO 2000
1990 DISPLAY AT(23,1):"all i can say is
practice,  practice, practice!!!"
2000 GOSUB 2830
2010 ACCEPT AT(1,21):Q$
2020 STOP
2030 FOR R1=-1 TO 1
2040 FOR C1=-1 TO 1
2050 IF A(R+R1,C+C1)=T2 THEN 2120
2060 NEXT C1
2070 NEXT R1
2080 REM NO T2 FOUND,FAILURE
2090 F1=0
2100 RETURN
2110 REM SUCESS
2120 F1=1
2130 RETURN
2140 REM
2150 S1=0
```

```
216Ø FOR K=1 TO 8
217Ø R5=R4(K)
218Ø C5=C4(K)
219Ø R6=R+R5
22ØØ C6=C+C5
221Ø S3=Ø
222Ø IF A(R6,C6)<>T2 THEN 24ØØ
223Ø REM LOOP THROUGH RUN
224Ø S3=S3+1
225Ø R6=R6+R5
226Ø C6=C6+C5
227Ø IF A(R6,C6)=T1 THEN 23ØØ
228Ø IF A(R6,C6)=Ø THEN 24ØØ
229Ø GOTO 224Ø
23ØØ S1=S1+S3
231Ø IF U<>1 THEN 24ØØ
232Ø REM UPDATE BOARD
233Ø R6=R
234Ø C6=C
235Ø FOR K1=Ø TO S3
236Ø A(R6,C6)=T1
237Ø R6=R6+R5
238Ø C6=C6+C5
239Ø NEXT K1
24ØØ NEXT K
241Ø RETURN
242Ø REM SUB PRINT BOARD
243Ø CALL CHAR(92,"ØØ7F7F7F7F7F7F7FØØFEF
EFEFEFEFEFE7F7F7F7F7F7F7F7FØØFEFEFEFEFEFEF
EØØ")
244Ø CALL CHAR(136,"ØØ7F7F7F7F7F7F7FØØFE
FEFEFEFEFEFE7F7F7F7F7F7F7FØØFEFEFEFEFEFE
FEØØ")
245Ø FOR I=2 TO 16 STEP 2
246Ø FOR J=2 TO 16 STEP 2
247Ø IF D$(A(I/2,J/2)+1)="X" THEN A1=92
248Ø IF D$(A(I/2,J/2)+1)="." THEN 254Ø
249Ø IF D$(A(I/2,J/2)+1)="O" THEN A1=136
25ØØ CALL HCHAR(I,3+J,A1)
251Ø CALL HCHAR(I,4+J,A1+1)
252Ø CALL HCHAR(1+I,3+J,A1+2)
253Ø CALL HCHAR(1+I,4+J,A1+3)
254Ø NEXT J
255Ø NEXT I
256Ø RETURN
```

```
257Ø REM DELETE SUB
258Ø DISPLAY AT(21,1):"":"":""
259Ø RETURN
26ØØ REM SET UP NEW GAME
261Ø FOR R=Ø TO 9
262Ø FOR C=Ø TO 9
263Ø A(R,C)=Ø
264Ø NEXT C :: NEXT R
265Ø RETURN
266Ø REM INITIAL BOARD
267Ø CALL CHAR(91,"FFFFFFFFFFFFFFFF")
268Ø CALL CHAR(133,"Ø1Ø1Ø1Ø1Ø1Ø1Ø1Ø1")
269Ø CALL CHAR(134,"ØØØØØØØØØØØØØØFF")
27ØØ CALL CHAR(135,"Ø1Ø1Ø1Ø1Ø1Ø1Ø1FF")
271Ø DISPLAY AT(1,3):" A B C D E F G H"
272Ø FOR ROW=2 TO 17 STEP 2
273Ø DISPLAY AT(ROW+1,1):ROW/2
274Ø FOR L=1 TO 15 STEP 2
275Ø CALL HCHAR(ROW,4+L,132,1)
276Ø CALL HCHAR(ROW,5+L,133,1)
277Ø CALL HCHAR(ROW+1,4+L,134,1)
278Ø CALL HCHAR(ROW+1,5+L,135,1)
279Ø NEXT L
28ØØ NEXT ROW
281Ø RETURN
282Ø REM TUNE
283Ø CALL SOUND(17Ø,33Ø,1)
284Ø CALL SOUND(17Ø,415,1)
285Ø CALL SOUND(17Ø,494,1)
286Ø CALL SOUND(17Ø,659,1)
287Ø CALL SOUND(8Ø,11Ø,3Ø)
288Ø CALL SOUND(17Ø,494,1)
289Ø CALL SOUND(4ØØ,659,1)
29ØØ RETURN
```

Extended Basic
Printer Optional

SEEK AND FIND GENERATOR

Brian Prothro

Puzzles have occupied our minds for centuries. Like many other popular games today we also find ourselves pitting our wits agenst computer created puzzles.

Here is a SEEK AND FIND puzzle generator that lets you customize your puzzles by allowing you to enter your own words (up to 35). You can send messages, quotes, birthday greetings or enter vocabulary to study. The possibilities are endless. You are limited only by your imagination.

Once entered, the generator embeds the word list within the puzzle while criss-crossing and overlapping some of the words with common letters. You can play the puzzle on the screen, or print copies with a printer. Be forewarned, the words appear within the puzzle written forward, backward, up, down and diagonally. Tough !

The list allows up to 35 words with a maximum of 1Ø letters each. Invalid words (including numbers or words that are too long) are not accepted and may be reentered or changed. Entering the number nine terminates the word entry mode and starts the puzzler working. Once having

entered the word list the program then asks if you want to watch where the words will appear as they are written into the puzzle. You should watch this a few times, it is fastinating to see how the puzzles are constructed.

When creating a word list be careful not to enter too many lengthy words at the end of a long list. The computer randomly tries to fit the words into the remaining spaces in the puzzle and toward the end it can be a tight fit. In this case the computer uses one hundred random tries in which to fit the word. Then it asks whether to delete the word or continue attempting insertion. You may continue the attempt as many times as desired. After the puzzle is generated you have the option of playing the screen or printing copies to the printer. Line 138Ø of the program contains the OPEN statement for the printer. You may need to modify this code to fit your printers parametes.

PLAYING THE SCREEN

Keys one through four move the cursor around the screen. After the cursor is located over a word, press ENTER to check your guess. The program keeps track of each correctly guessed word. When you press the S' key to scroll the word list you will notice the message "USED WORD" in place of each correctly guessed word. Once guessed, the words cannot be guessed again, and will give a "wrong guess" message.

CODE DESCRIPTION

After the word list is loaded into the array, one at a time the words are tested and placed into the puzzle array. Lines 35Ø-37Ø randomly select a starting point and direction for the placement of a word. Where the variable RD is a random direction, and R and C are a random starting row and column.

Lines 39Ø-58Ø check to see if the length of the word will exceed the puzzle boundries in the given direction. If the word fits then lines 59Ø-65Ø check each letter position for the word in the puzzle array to see if the position is occupied (Coincidental letters of equal value are acceptable, creating the possbility of overlapped words). If the space is not occupied (P$(R,C)=Ø) then lines 1Ø9Ø-118Ø continue to incriment in the direction chosen for the word untill the placement checkout is complete.

Assuming the puzzle positions for the word are all clear or acceptable, lines 68Ø-8ØØ then will save the letters of the word into the puzzles array (PUZ$(R,C)). At this point if the option to watch the puzzle be constructed was chosen by the user (SEE=1) then each letter will also appear on the screen as it is placed in the puzzle array.

When all the words are placed within the puzzle, lines 131Ø-136Ø fill the remaining empty positions of the puzzle array with ramdom letters. Due to the time consumed in generating a random character in line 133Ø, The puzzle fill time for this routine can take up to one minute. Although not included, this would be a good line to recode using a CALL PEEK to gain access to a random number. The speed gained would be worthwhile.

```
100 REM SEEK/FIND GENERATOR
110 REM CRACKING THE 99/4A
120 CALL CLEAR
130 DIM CHAR$(26),W$(35),PUZ$(20,20),P$(
20,20),Z$(35),RR(35),CC(35)
140 GOSUB 870
150 PRINT :,:,:,:,"PLEASE WAIT..."
160 GOSUB 960
170 CALL CLEAR
180 DISPLAY AT(10,4):"ENTER YOUR WORD LI
ST":"    ENTER 9 TO FINISH"
190 FOR I=1 TO 35
200 PRINT I
210 ACCEPT AT(23,5)SIZE(10)BEEP:W$(I)
220 IF W$(I)="9" THEN 250
230 GOSUB 1020
240 NEXT I
250 W$(I)="" :: TW,I=I-1
260 CALL CLEAR
270 GOSUB 1530
280 DISPLAY AT(23,4):W$(LOOP)
290 DISPLAY AT(23,1):"SEE WORD LOCATIONS
 (Y/N)"
300 ACCEPT AT(23,27)VALIDATE("YN")SIZE(1
)BEEP:Q$
310 DISPLAY AT(23,1):"   PLEASE WAIT..."
320 IF Q$="Y" THEN SEE=1 ELSE SEE=0
330 LOOP=LOOP+1
340 RANDOMIZE
350 RD=INT(RND*8)+1
360 R=INT(RND*20)+1
370 C=INT(RND*20)+1
380 ROW=R :: COL=C
390 REM CHOOSE DIRECTION
400 LN=LEN(W$(LOOP))
410 TRIES=TRIES+1
420 IF TRIES>100 THEN GOSUB 1200
430 ON RD GOTO 440,460,480,500,520,540,5
60,580
440 IF C+LN>20 THEN 340 ELSE GOSUB 600
450 GOTO 600
460 IF C-LN<0 THEN 340 ELSE GOSUB 600
470 GOTO 600
480 IF R+LN>20 THEN 340 ELSE GOSUB 600
490 GOTO 600
```

```
500 IF R-LN<0 THEN 340 ELSE GOSUB 600
510 GOTO 600
520 IF C-LN<1 OR R-LN<1 THEN 340 ELSE GO
SUB 600
530 GOTO 600
540 IF C+LN>20 OR R+LN>20 THEN 340 ELSE
GOSUB 600
550 GOTO 600
560 IF C-LN<1 OR R+LN>20 THEN 340 ELSE G
OSUB 600
570 GOTO 600
580 IF C+LN>20 OR R-LN<1 THEN 340 ELSE G
OSUB 600
590 REM LOCATION FREE?
600 FOR M=1 TO LEN(W$(LOOP))
610 IF R<1 OR R>20 OR C<1 OR C>20 THEN 3
40
620 IF P$(R,C)="0" THEN 640
630 IF PUZ$(R,C)=SEG$(W$(LOOP),M,1)THEN
640 ELSE 340
640 GOSUB 1090
650 NEXT M
660 R=ROW :: C=COL
670 REM PRINT WORD TO MEM.
680 RR(LOOP)=R :: CC(LOOP)=C
690 IF R<1 OR R>20 OR C<1 OR C>20 THEN 3
40
700 FOR L=1 TO LEN(W$(LOOP))
710 PUZ$(R,C)=SEG$(W$(LOOP),L,1)
720 P$(R,C)="1"
730 IF SEE=1 THEN 740 ELSE 760
740 D=ASC(PUZ$(R,C))
750 CALL HCHAR(R+1,C+5,D)
760 GOSUB 1090
770 NEXT L
780 IF LOOP=I THEN 800
790 GOTO 330
800 DISPLAY AT(23,1):"  THE WORDS ARE IN
 PLACE.":"I'M NOW FILLING THE SPACES."
810 GOSUB 1310
820 DISPLAY AT(23,1):"PRINT TO PRINTER?
(Y/N)":""
830 ACCEPT AT(23,26)SIZE(1)BEEP:Q$
840 IF Q$="Y" THEN 850 ELSE 1630
850 GOSUB 1380
```

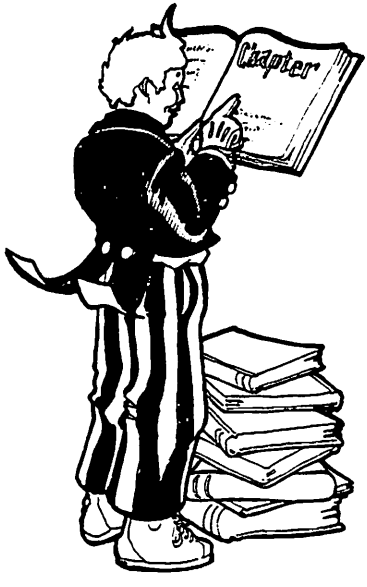
```
860 GOTO 820
870 FOR I=1 TO 226
880 DISPLAY AT(I/2+1,7):"SEEK AND FIND":
"       CUSTOMIZED":"     PUZZEL GENERAT
OR"
890 READ CHAR$(I)
900 DISPLAY AT(I/2+1,1):"":"":""
910 NEXT I
920 DATA A,B,C,D,E,F,G,H,I,J
930 DATA K,L,M,O,N,P,Q,R,S,T
940 DATA U,V,W,X,Y,Z
950 RETURN
960 FOR R=1 TO 20
970 FOR C=1 TO 20
980 P$(R,C)="0"
990 NEXT C :: NEXT R
1000 RETURN
1010 REM VALIDATE WORD ENTRY
1020 FOR J=1 TO LEN(W$(I))
1030 IF SEG$(W$(I),J,1)>"Z" OR SEG$(W$(I
),J,1)<"A" THEN 1040 ELSE 1060
1040 DISPLAY AT(23,1):"      invalid wor
d"
1050 I=I-1 :: GOTO 1070
1060 NEXT J
1070 RETURN
1080 REM DIRECTION
1090 ON RD GOTO 1100,1110,1120,1130,1140
,1150,1160,1170
1100 C=C+1 :: GOTO 1180
1110 C=C-1 :: GOTO 1180
1120 R=R+1 :: GOTO 1180
1130 R=R-1 :: GOTO 1180
1140 R=R-1 :: C=C-1 :: GOTO 1180
1150 R=R+1 :: C=C+1 :: GOTO 1180
1160 R=R+1 :: C=C-1 :: GOTO 1180
1170 R=R-1 :: C=C+1
1180 RETURN
1190 REM TROUBLE FITTING WORD
1200 TRIES=0
1210 DISPLAY AT(23,1):"STUCK ON WORD: ";
W$(LOOP):"DELETE IT? (Y/N)"
1220 ACCEPT AT(24,25)SIZE(1)BEEP:Q$
1230 DISPLAY AT(23,1):"":""
```

```
1240 DISPLAY AT(23,1):"    PLEASE WAIT..
."
1250 IF Q$="Y" THEN 1260 ELSE 1290
1260 TRIES=0
1270 W$(LOOP)="" :: DISPLAY AT(23,1):"CU
RRENT WORD DELETED":""
1280 TW=TW-1
1290 RETURN
1300 REM FILL BLANKS
1310 FOR R=1 TO 20
1320 FOR C=1 TO 20
1330 IF PUZ$(R,C)="" THEN PUZ$(R,C)=CHAR
$(INT(RND*26)+1)
1340 CALL HCHAR(R+1,C+5,ASC(PUZ$(R,C)))
1350 NEXT C :: NEXT R
1360 RETURN
1370 REM PRINT TO PRINTER
1380 OPEN #1:"RS232/1.BA=1200"
1390 FOR R=1 TO 20
1400 FOR C=1 TO 20
1410 LINE$=LINE$&PUZ$(R,C)&" "
1420 NEXT C
1430 PRINT #1:LINE$
1440 LINE$=""
1450 NEXT R
1460 PRINT #1
1470 FOR R=1 TO I STEP 2
1480 PRINT #1:W$(R),W$(R+1)
1490 NEXT R
1500 CLOSE #1
1510 RETURN
1520 REM DRAW BOX
1530 CALL CHAR(95,"FFFFFFFFFFFFFFFF")
1540 CALL CHAR(124,"FFFFFFFFFFFFFFFF")
1550 CALL HCHAR(1,5,95,22)
1560 CALL HCHAR(22,5,95,22)
1570 FOR J=1 TO 22
1580 CALL VCHAR(J,5,124)
1590 CALL VCHAR(J,26,124)
1600 NEXT J
1610 RETURN
1620 REM PLAY SCREEN
1630 DISPLAY AT(23,1):"P=PLAY SCREEN N=N
OT":""
```

```
1640 ACCEPT AT(23,28)VALIDATE("PN")BEEP:
Q$
1650 FOR TM=1 TO 150 :: NEXT TM
1660 IF Q$="P" THEN 1680
1670 IF Q$="N" THEN 820 ELSE 1630
1680 DISPLAY AT(23,1):"PRESS (ENTER) TO
GUESS      A WORD, S' TO SCROLL LIST."
1690 U,R,C=1
1700 CALL KEY(0,K,S)
1710 IF K=83 THEN GOSUB 2030
1720 CALL HCHAR(R+1,C+5,124)
1730 CALL HCHAR(R+1,C+5,ASC(PUZ$(R,C)))
1740 IF K=13 THEN GOSUB 1830
1750 IF K<49 OR K>52 THEN 1700
1760 RD=K-48 :: GOSUB 1090
1770 IF R<1 THEN R=1
1780 IF C<1 THEN C=1
1790 IF R>20 THEN R=19
1800 IF C>20 THEN C=20
1810 GOTO 1700
1820 REM ACCEPT WORD
1830 DISPLAY AT(23,1):"HOW LONG IS YOUR
WORD GUESS?":""
1840 ACCEPT AT(24,3)VALIDATE(NUMERIC)BEE
P:LENG
1850 IF LENG<1 OR LENG>10 THEN 1840
1860 FOR D=1 TO I
1870 IF R=RR(D)AND C=CC(D)AND LENG=LEN(W
$(D))THEN GUESS=1 ELSE 1890
1880 SS=D
1890 NEXT D
1900 IF GUESS=1 THEN 1920 ELSE 1990
1910 RETURN
1920 DISPLAY AT(23,1):"CORRECT GUESS"
1930 CALL SOUND(200,400,1,500,1):: CALL
SOUND(300,600,1,800,1)
1940 W$(SS)="used word!!"
1950 AL=AL+1
1960 IF AL=TW THEN DISPLAY AT(23,1):"YOU
 GOT THEM ALL !!!!!!!!" ELSE 1980
1970 GOTO 2070
1980 GUESS=0 :: RETURN
1990 DISPLAY AT(23,1):"WRONG, GUESS AGAI
N!!"
2000 CALL SOUND(100,137,1):: CALL SOUND(
```

```
420,127,1)
2010 RETURN
2020 REM SCROLL WORDS
2030 DISPLAY AT(23,1):W$(U);TAB(13);W$(U
+1):W$(U+2);TAB(13);W$(U+3)
2040 U=U+2
2050 IF U>I-2 THEN U=1
2060 RETURN
2070 CALL SOUND(200,247,1):: CALL SOUND(
200,330,1):: CALL SOUND(200,415,1):: CAL
L SOUND(200,494,1)
2080 CALL SOUND(40,300,30):: CALL SOUND(
200,414,1):: CALL SOUND(350,494,1)
2090 INPUT "HIT ENTER TO QUIT.":Q$
```

HOME



CHECK MANAGEMENT

Checkbook Management/Analysis is a set of two programs that allow the user to:

1. Balance his checkbook.
2. Save check information on a monthly basis (or on any interval desired).
3. Analyze each month's checks to determine: (1) the grand total of money spent using checks; (2) the number of checks and total spent in each of 20 categories (14 pre-defined and 6 user-defined); and (3) the percentage each category's total is of the grand total.
4. Maintain a cumulative record (in all categories as kept monthly), of where money has been spent, and analyze this cumulative record at any time. In effect, this shows the average spent in each category based on the number of months the cumulative file has been updated.

As stated, this is a set of two programs; they are called BALANCE and ANALYZE. BALANCE is run first and, if the user so chooses, BALANCE RUNs ANALYZE (using the Extended BASIC capablity of one program running another), so that they appear to the user as one program. However, it is possible to RUN ANALYZE by itself, but only after BALANCE has been run. You get exactly the sameresults.

INSTRUCTIONS

The balance program begins by asking you to enter, in the form mmyy, the month and year of the bank statement to be balanced. This creates the file CHECKSmmyy. If 9999 is entered for mmyy, BALANCE will create a data set called CHECKSTEMP when it is asked to save check information for analysis, regardless of whether a data set by this name already exists. The purpose of this feature is to analyze checks at any point during a pay period.

If you respond to mmyy with a date other than 9999, BALANCE examines the directory of the disk to ensure that a file for the month and year specified does not currently exist. Check information for a month is saved for ANALYZE in files with the general name format of CHECKSmmyy. These monthly files can be kept as long as needed, and analyzed at any time with ANALYZE.

After validating your entry, the balancing of the checkbook begins. The user enters information about (1) checks, (2) deposits, and (3) withdrawals (other than checks). For each of these three categories you can add, delete, and list entries. No matter what order the checks are entered, they are always listed in ascending numerical order. When finished entering all checkbook entries, the balance can be determined. At any time prior to terminating the program or invoking ANALYZE, you can make any corrections or changes required and re-calculate the balance.

You have the option to save the information and return later to complete the balancing activity. In this case, all input information, including deposits and withdrawals, are saved in a file with the general name SAVEmmyy. SAVEmmyy is deleted when BALANCE is terminated if the last save/fetch activity forSAVEmmyy was fetch.

When terminating BALANCE, the user is given three options: (1) saving check information on a file for analysis (CHECKSmmyy or CHECKSTEMP), (2) terminating the program, or (3) returning to the primary menu.

When (1) is selected the check file CHECKSmmyy is created if any month and year other than 9999 is specified; if 9999 was specified, the file created is named CHECKSTEMP. In addition, if a month and year other than 9999 is specified, another file, CMON, is created or updated, depending on whether or not it currently exists (BALANCE determines this by examining the disk directory). The CMON file contains one record: the mmyy as a string variable of the check file just created. This is to record the current month, or the most recent month balanced. This file is used by ANALYZE to determine the name of the data set to analyze when it is invoked by BALANCE or RUN independently.

If (1) is chosen, a menu appears giving the opportunity to terminate BALANCE with or without invoking ANALYZE.

If (2) is chosen, it means the user does not want the month analyzed and is using the system only for balancing purposes.

The analyze program can be run by selecting the "SAVE CHECKS TO FILE" option when terminating BALANCE or by RUNing it independently sometime after BALANCE. In either case, ANALYZE immediately accesses the CMON file and attempts to read the data set CHECKSmmyy, where the mmyy was obtained from the CMON file. The program will crash if the CMON file does not exist. In other words, ANALYZE will attempt to analyze the most recent month balanced when it is run without any menu appearing.

The report produced by ANALYZE consists of displaying on the screen all category names, the total number of checks written in each category, the total amount of money spent in each category, and the percentage of the month's total each category's total represents. In addition, displayed in the lower left-hand corner, there is the mmyy of the month being analyzed: "CUM" if you analyze the CUMULATIVE file, or TEMP if you are analyzing CHECKSTEMP.

If it is the first execution of ANALYZE, ANALYZE will create the CUMULATIVE file. After analyzing the current month, ANALYZE will attempt to integrate the current month's data into the CUMULATIVE file. If ANALYZE is RUN some other time after the current month is integrated, integration will be bypassed because ANALYZE checks the CINT file, which is updated by ANALYZE after an integration and tells the last month integrated.

After integration has occurred or been bypassed, as the case may be, a menu appears, giving you a choice between;

(1) analyzing the current (most recently balanced) month again.

(2) analyzing the CUMULATIVE file.

(3) analyzing any month the user chooses.

(4) analyzing the CHECKSTEMP file.

(5) terminating the program. If the month selected is not on the disk, an error message indicating this is displayed.

It would be best to put this program on a seperate disk in order to have room for your data files.

The following describes the files used in this program.

CMON is created and updated by the BALANCE, and interrogated by ANALYZE. This file holds one string variable indicating the last month and year balanced. The purpose of this file is to determine the proper month to analyze and to prevent ANALYZE from integrtating into the CUMULATIVE file the check data from the last month balanced.

CHECKSTEMP is created by BALANCE. If you enter 9999 as the month and year being balanced/analyzed, this indicates that you are entering only a portion of a months checks. This file can be analyzed from the menu and allows you to analyze the checkbook at anytime during the month.

CHECKSmmyy is created by BALANCE. If you specify any date other than 9999 as the month and year being balanced/analyzed. The mmyy of this data set is placed as a string variable into the CMON file, CMON is interrogated by ANALYZE when it is RUN in order to determine which month to analyze.

SAVEmmyy is created and retrieved by BALANCE when requested via menu. This file saves all information already input, including deposits and withdrawals. This is to be used if you are interrupted while balancing the checkbook and saves temporarily what you've done so that you can finish later.

CINT is created, interrogated, and updated by ANALYZE. This file tells ANALYZE the mmyy of the last month integrated into the CUMULATIVE file, which prevents integrating a month twice.

The BALANCE program is large and consequently, it will save to disk under the file attributes; BALANCE INT-VAR 254. This is peculiar to the 99/4A and creates no problems. The program can be loaded from disk with the usual OLD and RUN commands.

```
100 REM CHECKBOOK/MANAGEMENT
110 REM C 1983 BY JOHN COPE
120 REM CRACKING THE 99/4A
130 OPTION BASE 1
140 CALL CLEAR :: CALL SCREEN(4):: BC=33
 :: RP=21 :: CALL CHAR(BC,"FFFFFFFFFFFF
FFF"):: CALL COLOR(1,13,1)
150 DIM CKNO(101),CKAMT(101),CKCAT(101),
CKOUT(101),WITH(31),DEP(31),L$(18),MON$(
127):: MAXDEP,MAXWITH=30 :: MAXCK=101
160 CK=1 :: DP=1 :: WI=1
170 DISPLAY AT(4,1)ERASE ALL:"   WARNING
: DO NOT USE": :"   COMMAS WHEN ENTERING
": :"   AMOUNTS OF MONEY."
180 DISPLAY AT(15,1):"   SUCH ENTRIES WI
LL BE": :"   FOUND TO HAVE AN": :"   INV
ALID FORMAT." :: GOSUB 3110 :: GOSUB 347
0
190 DISPLAY AT(4,1)ERASE ALL:" ENTER THE
  MONTH AND  YEAR":" OF  THE STATEMENT
TO  BE":" BE BALANCED, IN THE":" FOLLOWI
NG FORMAT: MMYY"
200 DISPLAY AT(15,1):" OR ENTER 9999 FOR
 TEMPORARY":" CHECK ANALYSIS."
210 DISPLAY AT(9,1):" WHERE 'MM' = 01 FO
R JAN,": :" 11 = NOV, ETC, AND YY=YEAR."
 :: GOSUB 3110
220 ACCEPT AT(20,4)VALIDATE(DIGIT)SIZE(4
):MMYY
230 MY$=STR$(MMYY):: IF LEN(MY$)<3 THEN
220 :: IF LEN(MY$)=3 THEN MY$="0"&MY$
240 IF MMYY=9999 THEN MY$="TEMP" :: GOTO
 370
250 OPEN #33:"DSK1.",INPUT ,RELATIVE,INT
ERNAL
260 C=1 :: FOR I=1 TO 127 :: INPUT #33:A
$,A,J,K
270 IF LEN(A$)=0 THEN 300
```

```
280 MON$(C)=A$ :: C=C+1
290 NEXT I
300 CLOSE #33
310 DSN$="CHECKS"&MY$
320 FOR I=1 TO C :: IF DSN$=MON$(I)THEN
350
330 NEXT I
340 GOTO 370
350 DISPLAY AT(4,1)ERASE ALL:">> ERROR.
DISK  ALREADY HAS": :"   CHECKS FOR THE
MONTH": :"   AND YEAR ENTERED (";MY$;").
"
360 GOSUB 3110 :: GOSUB 3470 :: GOTO 190
370 REM CHECKBOOK BALANCING
380 DISPLAY AT(8,6)ERASE ALL:"BALANCE CH
ECKBOOK." :: GOSUB 3110 :: GOSUB 3470
390 DISPLAY AT(4,1)ERASE ALL:" *** BALAN
CE CHECKBOOK ***"
400 DISPLAY AT(6,1):"   *** PRIMARY MENU
 ***"
410 DISPLAY AT(8,1):" 1 - PROCESS A CHEC
K"
420 DISPLAY AT(10,1):" 2 - PROCESS A DEP
OSIT"
430 DISPLAY AT(12,1):" 3 - PROCESS A WIT
HDRAWAL"
440 DISPLAY AT(14,1):" 4 - CALCULATE BAL
ANCE"
450 DISPLAY AT(16,1):" 5 - TERMINATE BAL
ANCING"
460 DISPLAY AT(18,1):" 6 - SAVE CHECKBOO
K" :: DISPLAY AT(20,1):" 7 - FETCH CHECK
BOOK"
470 GOSUB 3500 :: GOSUB 3110 :: GOSUB 34
80
480 RV=RV-48 :: IF RV<1 THEN 470 :: IF R
V>7 THEN 470 :: ON RV GOSUB 500,1590,213
0,2630,2910,3140,3240
490 GOTO 390
500 REM PROCESS A CHECK
510 DISPLAY AT(4,1)ERASE ALL:" ** PROCES
S A CHECK **"
520 DISPLAY AT(7,1):" 1 - ADD A CHECK"
530 DISPLAY AT(9,1):" 2 - DELETE A CHECK
"
```

```
540 DISPLAY AT(11,1):" 3 - LIST THE CHEC
KS"
550 DISPLAY AT(13,1):" 4 - RETURN TO PRI
MARY MENU"
560 GOSUB 3500 :: GOSUB 3110 :: GOSUB 34
80
570 RV=RV-48 :: IF RV<1 THEN 560 :: IF R
V>4 THEN 560 :: ON RV GOSUB 590,1340,147
0,580
580 RETURN
590 REM ADD A CHECK
600 DISPLAY AT(3,1)ERASE ALL:" ENTER A C
HECK THIS WAY:"
610 DISPLAY AT(5,1):"   CKNO,AMOUNT,CATE
GORY[,1]"
620 DISPLAY AT(7,1):" WHERE ,1 IS CODED
ONLY WHEN THE CHECK IS OUTSTANDING."
630 DISPLAY AT(9,1):" VALID CATEGORY COD
ES:"
640 DISPLAY AT(11,1):"1=CASH     2=HOUSE
 3=CLOTHES":"4=MEDICAL 5=AUTO    6=CHARGE
":"7=FOOD     8=MISC    9=INSUR":"10=DONAT
E 11=UTILI 12=PHONE"
650 DISPLAY AT(15,1):"13=FUN     14-20 YO
U DEFINE":
660 DISPLAY AT(17,1):" PRESS <ENTER> TO
RETURN TO  THE PRIMARY MENU." :: GOSUB 3
110
670 IF CK=1 THEN 720 :: IF CK>MAXCK-1 TH
EN 3050
680 IF CKOUT(CK-1)=0 THEN 710
690 DISPLAY AT(20,1):"LAST: ";LCKNO :: D
ISPLAY AT(20,13):USING 3530:LCKAMT :: DI
SPLAY AT(20,23):LCKCAT;LCKOUT
700 GOTO 720
710 DISPLAY AT(20,1):"LAST: ";LCKNO :: D
ISPLAY AT(20,13):USING 3530:LCKAMT :: DI
SPLAY AT(20,23):LCKCAT
720 ACCEPT AT(23,5):CK$
730 IF CK$<>"" THEN 740 :: RETURN
740 GOSUB 980
750 IF VALRC<>0 THEN 590
760 REM CK FOR DUPLICATE
770 FOR I=1 TO CK-1
780 IF CKNO(I)=CKNO(CK)THEN 810
```

```
790 NEXT I
800 GOTO 870
810 DISPLAY AT(4,1)ERASE ALL:" >> ERROR.
 DUPLCATE CHECK        NUMBER."
820 DISPLAY AT(7,1):"     THE FOLLOWING C
HECK HAS      ALREADY BEEN ENTERED:"
830 DISPLAY AT(10,5):USING 3520:CKNO(I),
CKAMT(I),CKCAT(I)
840 DISPLAY AT(12,1):"     YOU ENTERED:"
850 DISPLAY AT(14,5):USING 3520:CKNO(CK)
,CKAMT(CK),CKCAT(CK)
860 GOSUB 3110 :: GOSUB 3470 :: GOTO 590
870 REM VALIDATE CAT CODE
880 IF CKCAT(CK)>20 THEN 960 ELSE CK=CK+
1 :: GOSUB 1330 :: IF CKNO(CK-1)>LGC THE
N LGC=CKNO(CK-1):: GOTO 670
890 FOR I=1 TO CK-2
900 IF CKNO(CK-1)<CKNO(I)THEN 930
910 NEXT I
920 DISPLAY AT(4,1)ERASE ALL:" >> ERROR.
 CANNOT PLACE": :"     CHECK NUMBER IN":
:"   SEQUENCE." :: GOSUB 3470 :: GOTO 30
90
930 SCK=CK :: FOR J=1 TO CK-I :: CKNO(CK
)=CKNO(CK-1):: CKAMT(CK)=CKAMT(CK-1):: C
KCAT(CK)=CKCAT(CK-1):: CKOUT(CK)=CKOUT(C
K-1):: CK=CK-1 :: NEXT J
940 CK=SCK :: CKNO(I)=CKNO(CK):: CKAMT(I
)=CKAMT(CK):: CKCAT(I)=CKCAT(CK):: CKOUT
(I)=CKOUT(CK)
950 CKNO(CK)=CKAMT(CK)=CKCAT(CK)=CKOUT(C
K)=0 :: GOTO 670
960 DISPLAY AT(4,1)ERASE ALL:" >> ERROR.
 INVALID CATEGORY      CODE";CKCAT(CK)
970 GOTO 860
980 REM VALIDATE FORMAT
990 IF LEN(CK$)>18 THEN 1290
1000 C1=POS(CK$,",",1):: IF C1=0 THEN 12
90
1010 IF C1>5 THEN 1290
1020 PP=POS(CK$,".",1)
1030 C2=POS(CK$,",",C1+1):: IF C2=0 THEN
 1290 :: IF LEN(CK$)=C2 THEN 1290
1040 IF PP=0 THEN 1050 ELSE IF C2-PP>3 T
HEN 1290
```

```
105Ø C3=POS(CK$,",",C2+1)
106Ø IF C3=Ø THEN CKOUT(CK)=Ø ELSE CKOUT
(CK)=1
107Ø IF C3=Ø THEN 109Ø
108Ø IF LEN(CK$)-C3<>1 THEN 129Ø :: IF S
EG$(CK$,LEN(CK$),1)<>"1" THEN 129Ø
109Ø IF C2=Ø THEN 129Ø
11ØØ IF C2-C1<=1 THEN 129Ø
1111Ø IF PP=Ø THEN 116Ø
1120 IF PP<C1 THEN 129Ø
1130 IF PP>C2 THEN 129Ø
1140 PP2=POS(CK$,".",PP+1)
115Ø IF PP2<>Ø THEN 129Ø
1160 FOR I=1 TO LEN(CK$)
1170 L$(I)=SEG$(CK$,I,1)
1180 NEXT I
1190 FOR I=1 TO LEN(CK$)
1200 IF ASC(L$(I))=44 THEN 123Ø :: IF AS
C(L$(I))=46 THEN 123Ø
1210 IF ASC(L$(I))<48 THEN 129Ø
1220 IF ASC(L$(I))>57 THEN 129Ø
1230 NEXT I :: CKNO(CK)=VAL(SEG$(CK$,1,C
1-1))
1240 CKAMT(CK)=VAL(SEG$(CK$,C1+1,C2-C1-1
))
125Ø IF C3=Ø THEN CKCAT(CK)=VAL(SEG$(CK$
,C2+1,LEN(CK$)-C2))
126Ø IF C3<>Ø THEN CKCAT(CK)=VAL(SEG$(CK
$,C2+1,C3-C2-1))
127Ø VALRC=Ø
128Ø RETURN
129Ø DISPLAY AT(4,1)ERASE ALL:" >> ERROR
. CHECK INFORMATION    NOT ENTERED IN TH
E          CORRECT FORMAT."
13ØØ DISPLAY AT(12,1):"    YOU ENTERED":
 :"      ";CK$
1310 GOSUB 3110 :: GOSUB 347Ø
1320 VALRC=4 :: GOTO 128Ø
1330 LCKNO=CKNO(CK-1):: LCKAMT=CKAMT(CK-
1):: LCKCAT=CKCAT(CK-1):: LCKOUT=CKOUT(C
K-1):: RETURN
1340 REM DELETE A CHECK
135Ø DISPLAY AT(4,1)ERASE ALL:"ENTER THE
 NUMBER OF THE": :"CHECK TO BE DELETED."
1360 GOSUB 3110 :: ACCEPT AT(10,5)VALIDA
```

```
TE(DIGIT)SIZE(4):A$ :: IF A$="" THEN 370
 ELSE A=VAL(A$)
1370 FOR I=1 TO CK-1
1380 IF A=CKNO(I)THEN 1420
1390 NEXT I
1400 DISPLAY AT(4,1)ERASE ALL:" >> ERROR
. CHECK NUMBER        ":"   ";A;"    NOT
 FOUND."
1410 GOSUB 3110 :: GOSUB 3470 :: GOTO 50
0
1420 FOR J=I+1 TO CK
1430 CKNO(J-1)=CKNO(J):: CKAMT(J-1)=CKAM
T(J):: CKCAT(J-1)=CKCAT(J)
1440 NEXT J
1450 CKNO(CK)=0 :: CKAMT(CK)=0 :: CKCAT(
CK)=0 :: CK=CK-1 :: IF A=LGC THEN LGC=CK
NO(CK-1)
1460 GOTO 500
1470 REM LIST THE CHECKS
1480 IF CK=1 THEN CALL CLEAR :: GOTO 158
0
1490 I=1
1500 CALL CLEAR :: FOR J=1 TO 15
1510 IF CKOUT(I)=0 THEN 1540
1520 DISPLAY AT(J+4,1):USING "####":CKNO
(I):: DISPLAY AT(J+4,8):USING 3530:CKAMT
(I):: DISPLAY AT(J+4,19):CKCAT(I):: DISP
LAY AT(J+4,24):CKOUT(I)
1530 GOTO 1550
1540 DISPLAY AT(J+4,1):USING "####":CKNO
(I):: DISPLAY AT(J+4,8):USING 3530:CKAMT
(I):: DISPLAY AT(J+4,19):CKCAT(I)
1550 I=I+1 :: IF I=CK THEN 1580
1560 NEXT J :: PRINT :: GOSUB 3110 :: GO
SUB 3470 :: GOTO 1500
1570 CALL CLEAR
1580 GOSUB 3510 :: GOSUB 3110 :: GOSUB 3
480 :: GOTO 500
1590 REM PROCESS A DEPOSIT
1600 TYPE$=" DEPOSIT "
1610 DISPLAY AT(4,1)ERASE ALL:"  * *  PR
OCESS A DEPOSIT * *"
1620 DISPLAY AT(7,1):" 1 - ADD IN A DEPO
SIT"
```

```
163Ø DISPLAY AT(9,1):" 2 - DELETE A DEPO
SIT"
164Ø DISPLAY AT(11,1):" 3 - LIST DEPOSIT
S ENTERED"
165Ø DISPLAY AT(13,1):" 4 - RETURN TO PR
IMARY MENU"
166Ø GOSUB 35ØØ :: GOSUB 311Ø :: GOSUB 3
48Ø
167Ø RV=RV-48 :: IF RV<=Ø THEN 166Ø :: I
F RV>4 THEN 166Ø :: ON RV GOSUB 169Ø,186
Ø,2Ø6Ø,168Ø
168Ø RETURN
169Ø REM ADD A DEPOSIT
17ØØ GOSUB 343Ø
171Ø IF DP=1 THEN 173Ø :: IF DP>MAXDEP T
HEN 3Ø7Ø
172Ø DISPLAY AT(1Ø,1):"LAST ENTERED:" ::
 DISPLAY AT(1Ø,16):USING 353Ø:DEP(DP-1)
173Ø GOSUB 344Ø
174Ø IF D$<>"" THEN 175Ø :: RETURN
175Ø GOSUB 176Ø :: IF RC<>Ø THEN 184Ø EL
SE 182Ø
176Ø P=POS(D$,",",1):: IF P<>Ø THEN 183Ø
177Ø FOR I=1 TO LEN(D$)
178Ø IF ASC(SEG$(D$,I,1))<48 THEN 179Ø :
: IF ASC(SEG$(D$,I,1))>57 THEN 183Ø :: G
OTO 18ØØ
179Ø P=POS(D$,".",1):: IF P=Ø THEN 183Ø
:: P1=POS(D$,".",P+1):: IF P1<>Ø THEN 18
3Ø :: IF LEN(D$)-P>2 THEN 183Ø
18ØØ NEXT I :: RC=Ø
181Ø RETURN
182Ø DEP(DP)=VAL(D$):: DP=DP+1 :: GOTO 1
7ØØ
183Ø RC=4 :: GOTO 181Ø
184Ø GOSUB 34ØØ
185Ø GOSUB 311Ø :: GOSUB 347Ø :: GOTO 16
9Ø
186Ø REM DELETE DEPOSIT
187Ø GOSUB 345Ø
188Ø IF D$="" THEN 191Ø
189Ø GOSUB 176Ø
19ØØ IF RC<>Ø THEN 198Ø ELSE 192Ø
191Ø RETURN
192Ø FOR I=1 TO DP-1
```

```
1930 IF VAL(D$)=DEP(I)THEN 2010
1940 NEXT I
1950 GOSUB 3410
1960 GOSUB 3110 :: GOSUB 3470
1970 RETURN
1980 GOSUB 3400
1990 GOSUB 3110 :: GOSUB 3470
2000 GOTO 1970
2010 FOR J=I TO DP-1
2020 DEP(J)=DEP(J+1)
2030 NEXT J
2040 DEP(DP-1)=0 :: DP=DP-1
2050 GOTO 1970
2060 REM LIST DEPOSITS
2070 CALL CLEAR
2080 FOR I=1 TO DP-1
2090 IF DEP(I)=0 THEN 32767
2100 DISPLAY AT(I+3,10):USING 3530:DEP(I
)
2110 NEXT I
2120 GOSUB 3510 :: GOSUB 3110 :: GOSUB 3
480 :: RETURN
2130 REM WITHDRAWALS
2140 TYPE$=" WITHDRAWAL "
2150 DISPLAY AT(4,3)ERASE ALL:"** PROCES
S WITHDRAWALS **"
2160 DISPLAY AT(7,2):"1 - ENTER A WITHDR
AWAL"
2170 DISPLAY AT(9,2):"2 - DELETE A WITHD
RAWAL"
2180 DISPLAY AT(11,2):"3 - LIST WITHDRAW
ALS"
2190 DISPLAY AT(13,2):"4 - RETURN TO PRI
MARY MENU"
2200 GOSUB 3500 :: GOSUB 3110 :: GOSUB 3
480
2210 IF RV<49 THEN 2200
2220 IF RV>52 THEN 2200
2230 RV=RV-48 :: ON RV GOSUB 2250,2460,2
390,2240
2240 RETURN
2250 REM ENTER WITHDRAWAL
2260 GOSUB 3430
2270 IF WI=1 THEN 2290 :: IF WI>MAXWITH
THEN 3060
```

```
2280 DISPLAY AT(10,1):"LAST ENTERED:" ::
 DISPLAY AT(10,15):USING 3530:WITH(WI-1)
2290 GOSUB 3440
2300 IF D$="" THEN 2310 ELSE 2320
2310 RETURN
2320 GOSUB 1760
2330 IF RC<>0 THEN 2360
2340 WITH(WI)=VAL(D$)
2350 WI=WI+1 :: GOTO 2270
2360 GOSUB 3400
2370 GOSUB 3110 :: GOSUB 3470
2380 GOTO 2130
2390 REM LIST WITHDRAWALS
2400 CALL CLEAR
2410 FOR I=1 TO WI-1
2420 IF WITH(I)=0 THEN 32767
2430 DISPLAY AT(6+I,10):USING 3530:WITH(
I)
2440 NEXT I
2450 GOSUB 3510 :: GOSUB 3110 :: GOSUB 3
480 :: RETURN
2460 REM DELETE A WITH.
2470 GOSUB 3450
2480 IF D$<>"" THEN 2490 :: RETURN
2490 GOSUB 1760
2500 IF RC<>0 THEN 2560
2510 FOR I=1 TO WI-1
2520 IF WITH(I)=VAL(D$)THEN 2570
2530 NEXT I
2540 GOSUB 3410
2550 GOSUB 3110 :: GOSUB 3470 :: GOTO 22
40
2560 GOSUB 3400 :: GOSUB 3470 :: GOTO 26
20
2570 FOR J=I TO WI-1
2580 WITH(J)=WITH(J+1)
2590 NEXT J
2600 IF WI=1 THEN 2620
2610 WITH(WI-1)=0 :: WI=WI-1
2620 RETURN
2630 REM FIND BALANCE
2640 SCK,SDP,SWI,SOUT=0
2650 DISPLAY AT(4,1)ERASE ALL:"ENTER BAL
ANCE FROM PREVIOUS": :"MONTH:" :: GOSUB
```

```
311Ø :: ACCEPT AT(8,5)VALIDATE(DIGIT,"."
):D$
266Ø IF D$="" THEN 37Ø
267Ø GOSUB 176Ø :: IF RC<>Ø THEN 263Ø
268Ø FOR I=1 TO CK-1
269Ø SCK=SCK+CKAMT(I)
27ØØ NEXT I
271Ø FOR I=1 TO DP-1
272Ø SDP=SDP+DEP(I)
273Ø NEXT I
274Ø FOR I=1 TO WI-1
275Ø SWI=SWI+WITH(I)
276Ø NEXT I
277Ø NOUT=Ø
278Ø FOR I=1 TO CK-1
279Ø IF CKOUT(I)=Ø THEN 282Ø
28ØØ NOUT=NOUT+1
281Ø SOUT=SOUT+CKAMT(I)
282Ø NEXT I
283Ø BAL=VAL(D$)-SCK-SWI+SDP
284Ø DISPLAY AT(4,1)ERASE ALL:" ITEM" ::
 DISPLAY AT(4,11):"NUMBER" :: DISPLAY AT
(4,24):"SUM"
285Ø DISPLAY AT(8,1):"PRIOR BAL." :: DIS
PLAY AT(8,2Ø):USING 353Ø:VAL(D$)
286Ø DISPLAY AT(1Ø,1):"CHECKS" :: DISPLA
Y AT(1Ø,13):CK-1 :: DISPLAY AT(1Ø,2Ø):US
ING 353Ø:SCK
287Ø DISPLAY AT(12,1):"DEPOSITS" :: DISP
LAY AT(12,13):DP-1 :: DISPLAY AT(12,2Ø):
USING 353Ø:SDP
288Ø DISPLAY AT(14,1):"WITHDRAWALS" :: D
ISPLAY AT(14,13):WI-1 :: DISPLAY AT(14,2
Ø):USING 353Ø:SWI
289Ø DISPLAY AT(16,1):"OUT CKS." :: DISP
LAY AT(16,13):NOUT :: DISPLAY AT(16,2Ø):
USING 353Ø:SOUT
29ØØ DISPLAY AT(2Ø,5):"BALANCE = " :: DI
SPLAY AT(2Ø,15):USING 353Ø:BAL :: GOSUB
311Ø :: GOSUB 347Ø :: RETURN
291Ø REM WRITE NEW ONES
292Ø DISPLAY AT(4,1)ERASE ALL:" ** TERMI
NATING MENU **": : :" 1 - SAVE CHECKS FO
R":"     ANALYSIS": :" 2 - TERMINATE PRO
GRAM"
```

```
2930 DISPLAY AT(12,1):" 3 - RETURN TO PR
IMARY":"      MENU" :: GOSUB 3500
2940 GOSUB 3110 :: GOSUB 3480 :: RV=RV-4
8 :: IF RV<0 THEN 2940 :: IF RV>3 THEN 2
940 :: ON RV GOTO 2950,3090,370
2950 OPEN #2:"DSK1.CHECKS"&MY$,SEQUENTIA
L,OUTPUT,INTERNAL,VARIABLE 64
2960 FOR I=1 TO CK-1
2970 PRINT #2:CKNO(I),CKAMT(I),CKCAT(I)
2980 NEXT I
2990 CLOSE #2
3000 IF MY$="TEMP" THEN 3020
3010 OPEN #1:"DSK1.CMON",OUTPUT,DISPLAY
,FIXED 10 :: PRINT #1:MY$ :: CLOSE #1
3020 DISPLAY AT(4,5)ERASE ALL:" * *  EXI
T MENU  * *"
3030 DISPLAY AT(9,1):" 1 - RUN ANALYSIS"
: :" 2 - TERMINATE PROGRAM" :: GOSUB 350
0 :: GOSUB 3110 :: GOSUB 3480 :: RV=RV-4
8 :: IF RV>2 THEN 3030
3040 IF RV=1 THEN DELETE "DSK1.SAVE"&MY$
 :: RUN "DSK1.ANALYZE" ELSE 3090
3050 DISPLAY AT(4,1)ERASE ALL:"MAXIMUM O
F 70 CHECKS HAS": :"BEEN REACHED." :: GO
SUB 3080 :: GOSUB 3110 :: GOSUB 3470 ::
RETURN
3060 DISPLAY AT(4,1)ERASE ALL:"THE MAXIM
UM OF 10 WITH-": :"DRAWALS HAS BEEN REAC
HED." :: GOSUB 3080 :: GOSUB 3110 :: GOS
UB 3470 :: RETURN
3070 DISPLAY AT(4,1)ERASE ALL:"THE MAXIM
UM OF 10 DEPOSITS": :"HAS BEEN REACHED."
 :: GOSUB 3080 :: GOSUB 3110 :: GOSUB 34
70 :: RETURN
3080 DISPLAY AT(10,1):"CALCULATE BALANCE
, SAVE": :"CHECKS, AND START PROGRAM": :
"AGAIN." :: RETURN
3090 IF FUNC$="FETCH" THEN DELETE "DSK1.
SAVE"&MY$
3100 CALL CLEAR :: END
3110 REM MAKE BORDER
3120 CALL COLOR(1,1,1):: CALL VCHAR(1,31
,BC,RP):: CALL VCHAR(1,32,BC,RP):: CALL
VCHAR(1,1,BC,RP):: CALL VCHAR(1,2,BC,RP)
3130 CALL HCHAR(1,1,BC,31):: CALL HCHAR(
```

```
2,1,BC,31):: CALL HCHAR(22,1,BC,32):: CA
LL COLOR(1,13,1):: RETURN
3140 REM SAVE CHECKBOOK
3150 FUNC$="SAVE"
3160 OPEN #44:"DSK1.SAVE"&MY$,OUTPUT,INT
ERNAL,SEQUENTIAL,VARIABLE 64
3170 FOR I=1 TO CK-1
3180 PRINT #44:CKNO(I),CKAMT(I),CKCAT(I)
,CKOUT(I):: NEXT I :: PRINT #44:99999,99
999,99999,99999
3190 FOR I=1 TO DP-1
3200 PRINT #44:DEP(I):: NEXT I :: PRINT
#44:99999
3210 FOR I=1 TO WI-1
3220 PRINT #44:WITH(I):: NEXT I :: PRINT
 #44:99999
3230 PRINT #44:LGC,LCKNO,LCKAMT,LCKCAT,L
CKOUT :: CLOSE #44 :: RETURN
3240 REM FETCH CHECKBOOK
3250 FUNC$="FETCH"
3260 OPEN #44:"DSK1.SAVE"&MY$,INPUT ,INT
ERNAL,SEQUENTIAL,VARIABLE 64
3270 FOR I=1 TO MAXCK
3280 INPUT #44:CKNO(I),CKAMT(I),CKCAT(I)
,CKOUT(I):: IF CKCAT(I)=99999 THEN 3300
3290 NEXT I
3300 CK=I
3310 FOR I=1 TO MAXDEP
3320 INPUT #44:DEP(I):: IF DEP(I)=99999
THEN 3340
3330 NEXT I
3340 DP=I
3350 FOR I=1 TO MAXWITH
3360 INPUT #44:WITH(I):: IF WITH(I)=9999
9 THEN 3380
3370 NEXT I
3380 WI=I
3390 INPUT #44:LGC,LCKNO,LCKAMT,LCKCAT,L
CKCAT :: CLOSE #44 :: RETURN
3400 DISPLAY AT(4,1)ERASE ALL:">> ERROR.
";TYPE$;"ENTERED": :"   IN INVALID FORMA
T." :: GOSUB 3420 :: RETURN
3410 DISPLAY AT(4,1)ERASE ALL:">> ERROR.
";TYPE$;"ENTERED": :"   NOT FOUND." :: G
```

```
OSUB 342Ø :: RETURN
342Ø DISPLAY AT(1Ø,4):"YOU ENTERED:" ::
DISPLAY AT(13,8):D$ :: RETURN
343Ø DISPLAY AT(4,1)ERASE ALL:"ENTER AMO
UNT OF";TYPE$;".": : : :"PRESS <ENTER> W
HEN NO MORE." :: RETURN
344Ø DISPLAY AT(18,1):"AMOUNT:" :: GOSUB
 311Ø :: ACCEPT AT(18,12)VALIDATE(DIGIT,
"."):D$ :: RETURN
345Ø DISPLAY AT(4,1)ERASE ALL:" ENTER AM
OUNT OF": :TYPE$;"TO BE DELETED.": : : :
" PRESS <ENTER> IF NO MORE."
346Ø GOSUB 311Ø :: ACCEPT AT(18,8)VALIDA
TE(DIGIT,"."):D$ :: RETURN
347Ø DISPLAY AT(23,1):"   >>> PRESS ANY
KEY":"       TO CONTINUE."
348Ø CALL KEY(Ø,RV,SV)
349Ø IF SV=Ø THEN 348Ø :: RETURN
35ØØ DISPLAY AT(23,3):">> PRESS NUMBER F
OR":"     FUNCTION DESIRED" :: RETURN
351Ø DISPLAY AT(24,4):"   **  LISTED  **
" :: RETURN
352Ø IMAGE #### #####.## ##
353Ø IMAGE #####.##
```

ANALYSIS

PROGRAM

```
1ØØ CALL SCREEN(4):: BC=33 :: CALL CHAR(
BC,"FFFFFFFFFFFFFFFF"):: CALL CLEAR
11Ø CALL CHARPAT(37,PCEN$,36,DLR$):: CAL
L CHAR(91,PCEN$,93,DLR$):: GOTO 45Ø
12Ø REM CHECKBOOK MGMT
13Ø REM UTILITY/ANALYSIS
14Ø REM C JOHN COPE 1983
15Ø REM CRACKING THE 99/4A
16Ø OPTION BASE 1
17Ø DIM T(2Ø),P(2Ø),SC(2Ø),CT(2Ø),CSC(2Ø
),MON$(127)
```

```
180 DISPLAY AT(4,1)ERASE ALL:"  *** PRIM
ARY MENU ***"
190 DISPLAY AT(6,1):" 1 - ANALYZE ANY MO
NTH"
200 DISPLAY AT(8,1):" 2 - ANALYZE CURREN
T MONTH"
210 DISPLAY AT(10,1):" 3 - ANALYZE CUMUL
ATIVE"
220 DISPLAY AT(12,1):" 4 - ANALYZE TEMP.
 CHECKS": :" 5 - TERMINATE PROGRAM"
230 DISPLAY AT(23,1):"  PRESS NUMBER FOR
 FUNCTION   DESIRED."
240 GOSUB 1180 :: GOSUB 1250 :: IF RV>5
THEN 240
250 ON RV GOTO 270,450,1040,260,1290
260 MY$="TEMP"
270 REM ANALYZE ANY MONTH
280 FOR I=1 TO 20 :: T(I)=0 :: SC(I)=0 :
: NEXT I :: S=0 :: IF MY$="TEMP" THEN GO
SUB 320 :: GOTO 390
290 DISPLAY AT(4,1)ERASE ALL:"ENTER MONT
H AND YEAR": :"OF MONTH YOU WANT TO": :"
ANALYZE, IN THE FOLLOWING": :"FORMAT:"
300 DISPLAY AT(12,1):"   MMYY": : :"WHER
E 'MM' IS THE MONTH": :"(E.G., 02=FEB, 1
1=NOV)": :"AND YY = THE YEAR."
310 GOSUB 1180 :: ACCEPT AT(23,5)VALIDAT
E(DIGIT)SIZE(4):MY :: MY$=STR$(MY):: IF
LEN(MY$)<3 THEN 310 :: IF LEN(MY$)=3 THE
N MY$="0"&MY$ :: GOSUB 320 :: GOTO 390
320 OPEN #32:"DSK1.",INPUT ,RELATIVE,INT
ERNAL
330 C=1 :: FOR I=1 TO 127
340 INPUT #32:A$,A,J,K
350 IF LEN(A$)=0 THEN 380
360 MON$(C)=A$ :: C=C+1
370 NEXT I
380 CLOSE #32 :: RETURN
390 DSN$="CHECKS"&MY$
400 FOR I=1 TO 24
410 IF DSN$=MON$(I)THEN 480
420 NEXT I
430 DISPLAY AT(4,1)ERASE ALL:">> ERROR.
FILE DOES NOT": :"   EXIST FOR THE MONTH
```

```
": :"   AND YEAR YOU CHOSE": :"   (";MY$
;")."
440 GOSUB 1180 :: GOSUB 1240 :: GOTO 120
450 REM ANALYZE THIS MO.
460 FOR I=1 TO 20 :: T(I)=0 :: SC(I)=0 :
: NEXT I :: S=0
470 OPEN #33:"DSK1.CMON",INPUT ,DISPLAY
,FIXED 10 :: INPUT #33:MY$ :: CLOSE #33
480 CALL CLEAR
490 OPEN #1:"DSK1.CHECKS"&MY$,SEQUENTIAL
,INTERNAL,INPUT ,VARIABLE 64
500 INPUT #1:A,B,C
510 GOTO 540
520 CLOSE #1
530 GOTO 550
540 S=S+B :: T(C)=T(C)+1 :: SC(C)=SC(C)+
B :: IF EOF(1)<>0 THEN 520 :: GOTO 500
550 FOR I=1 TO 20
560 P(I)=(SC(I)/S)*100
570 NEXT I
580 GOSUB 590 :: GOSUB 620 :: GOTO 120
590 DISPLAY AT(1,1)ERASE ALL:" CAT.   NO
.  PER     TOTAL"
600 DISPLAY AT(3,1):"CASH":"HOUSE":"CLOT
HES":"MEDICAL":"AUTO":"CHARGE":"FOOD":"M
ISC":"INSUR":"DONATE":"UTILI":"PHONE":"F
UN":"YOUR 14"
610 DISPLAY AT(17,1):"YOUR 15":"YOUR 16"
:"YOUR 17":"YOUR 18":"YOUR 19":"YOUR 20"
 :: RETURN
620 FOR I=1 TO 20
630 DISPLAY AT(2+I,9):USING "## ##.##[
]#####.##":T(I),P(I),SC(I)
640 NEXT I
650 DISPLAY AT(24,5):USING 1300:S
660 DISPLAY AT(24,1)SIZE(4):MY$
670 CALL KEY(0,RVV,SV):: IF SV=0 THEN 67
0
680 GOTO 690 !GO INTEGRATE
690 REM COMBINE RESULTS
700 IF INTG=1 THEN 120 :: INTG=1 :: CALL
 CLEAR :: DISPLAY AT(4,1)ERASE ALL:"  IN
TEGRATING ... PLEASE": :"  WAIT"
710 GOSUB 320
720 FOR I=1 TO C
```

```
730 IF MON$(I)="CINT" THEN 760
740 NEXT I
750 OPEN #25:"DSK1.CINT",OUTPUT,DISPLAY
,FIXED 10 :: PRINT #25:MY$ :: CLOSE #25
:: GOTO 770
760 OPEN #25:"DSK1.CINT",INPUT ,DISPLAY
,FIXED 10 :: INPUT #25:CINT$ :: CLOSE #2
5 :: IF CINT$=MY$ THEN GOSUB 1210 :: GOT
O 120 ELSE 750
770 OPEN #32:"DSK1.",INPUT ,RELATIVE,INT
ERNAL
780 FOR I=1 TO 127 :: INPUT #32:A$,A,J,K
 :: IF LEN(A$)=0 THEN ANS$="N" :: GOTO 8
10
790 IF A$="CUMULATIVE" THEN ANS$="Y" ::
GOTO 810
800 NEXT I :: ANS$="N"
810 CLOSE #32
820 IF ANS$="Y" THEN 890
830 OPEN #1:"DSK1.CUMULATIVE",INTERNAL,O
UTPUT,SEQUENTIAL,FIXED 50
840 FOR I=1 TO 20
850 PRINT #1:T(I),SC(I)
860 NEXT I
870 CLOSE #1
880 GOTO 120
890 OPEN #1:"DSK1.CUMULATIVE",INTERNAL,S
EQUENTIAL,UPDATE,FIXED 50
900 FOR I=1 TO 20 :: CT(I)=0 :: CSC(I)=0
 :: NEXT I
910 FOR I=1 TO 20
920 INPUT #1:CT(I),CSC(I)
930 NEXT I
940 FOR I=1 TO 20
950 CT(I)=CT(I)+T(I)
960 CSC(I)=CSC(I)+SC(I)
970 NEXT I
980 RESTORE #1
990 FOR I=1 TO 20
1000 PRINT #1:CT(I),CSC(I)
1010 NEXT I
1020 CLOSE #1
1030 GOTO 120
1040 REM ANALYZE CUMULATIVE
1050 FOR I=1 TO 20 :: SC(I)=0 :: T(I)=0
```

```
:: NEXT I :: S=Ø
1Ø6Ø OPEN #1:"DSK1.CUMULATIVE",INTERNAL,
SEQUENTIAL,INPUT ,FIXED 5Ø
1Ø7Ø FOR I=1 TO 2Ø
1Ø8Ø INPUT #1:T(I),SC(I)
1Ø9Ø NEXT I
11ØØ CLOSE #1
111Ø FOR I=1 TO 2Ø
112Ø S=S+SC(I)
113Ø NEXT I
114Ø FOR I=1 TO 2Ø
115Ø P(I)=(SC(I)/S)*1ØØ
116Ø NEXT I
117Ø MY$="CUM " :: GOSUB 59Ø :: GOSUB 62
Ø :: GOSUB 125Ø :: GOTO 12Ø
118Ø REM MAKE BORDER
119Ø CALL COLOR(1,1,1):: CALL VCHAR(1,1,
BC,24):: CALL VCHAR(1,2,BC,24):: CALL VC
HAR(1,31,BC,24):: CALL VCHAR(1,32,BC,24)
12ØØ CALL HCHAR(1,1,BC,31):: CALL HCHAR(
2,1,BC,31):: CALL HCHAR(22,1,BC,31):: CA
LL COLOR(1,13,1):: RETURN
121Ø DISPLAY AT(1Ø,1):"  INTEGRATION BYP
ASSED - ": :"  CURRENT MONTH ALREADY": :
"  INTEGRATED."
122Ø GOSUB 118Ø
123Ø FOR I=1 TO 2ØØØ :: NEXT I :: RETURN
124Ø DISPLAY AT(23,1):" PRESS ANY KEY TO
 CONTINUE"
125Ø CALL KEY(Ø,RV,SV)
126Ø IF SV=Ø THEN 125Ø
127Ø RV=RV-48
128Ø RETURN
129Ø CALL CLEAR :: END
13ØØ IMAGE " GRAND TOTAL = ]#####.##"
```

SUPER CATALOGGER

As you've probably found, there are a number of catalog listers available. Often you will find that a program, once found in the listing, then must be loaded and run. This program overcomes this inconvenience and offers a few additional options.

When running the program, the first prompt to appear asks whether you would like a copy of the catalog sent to the printer. The letter "N" for NO is the default that appears under the cursor, and may be selected simply by pressing enter. Otherwise pressing "Y" for YES will lead you through a series of parameters such as the RS232 port number you are using, the number of bits used (7 or 8), and the baud rate. Again, the letter the cursor flashes above will be the default value if you press the enter key.

Following the printer option, the program loads all program names, file sizes, and file types into an array and displays the directory on the screen in one shot. (Listings longer than one screen may be continued by pressing <ENTER>. Any program may be chosen and RUN by entering the number that appears next to the program name. What a time saver!

```
100 REM CRACKING THE 99/4A
110 REM SUPERCATALOGGER
120 REM BRUCE WYCHE 02/08/84
130 ON BREAK NEXT :: ON WARNING NEXT ::
ON ERROR 700
140 UNTRACE :: CALL CHARSET
150 DIM AA(127)
160 DIM B$(5),O$(127),G$(127),TP$(5)
170 BL$=RPT$(" ",56):: DEC$="0123456789"
180 CALL CLEAR :: CALL PEEK(-24576,@)::
IF @<>0 THEN CALL INIT :: CALL LOAD(-318
06,16):: CALL LOAD(-31878,0)
190 RESTORE 640 :: FOR I=1 TO 5 :: READ
```

```
B$(I),TP$(I):: NEXT I


210 DISPLAY ERASE ALL AT(2,1):"  SUPERCA
TALOGGER PROGRAM"
220 DISPLAY AT(4,1):"      READ OR PRINT
OUT"
230 DISPLAY AT(6,1):"    AUTOMATIC RUN O
PTION"
240 RESTORE 650 :: NK=1 :: GOSUB 610 ::
IF A$(1)="N" THEN 270
250 NK=3 :: GOSUB 610
260 PF=-1 :: OPEN #2:"RS232/"&A$(1)&".DA
="&A$(2)&".BA="&B$(VAL(A$(3)))
270 DISPLAY AT(20,1):"READING CATALOG, S
TANDBY...."
280 OPEN #1:"DSK1.",INPUT ,RELATIVE,INTE
RNAL :: INPUT #1:@$,@,J,K
290 H$="DSK."&@$&" "&"AVAIL "&STR$(K)&"/
"&STR$(J)
300 H$=H$&RPT$(" ",ABS((LEN(H$)<29))*(28
-LEN(H$)))&"NM FILENAME  SIZ TYPE  LEN P
"
310 I=0
320 I=I+1 :: INPUT #1:G$(I),AA(I),J,K ::
 IF LEN(G$(I))=0 THEN LF=I-1 :: CLOSE #1
 :: GOTO 360
330 O$(I)=STR$(I)&RPT$(" ",3-LEN(STR$(I)
))&G$(I)&RPT$("-",INT(10-LEN(G$(I))))&ST
R$(J)&RPT$(" ",3-LEN(STR$(J)))&TP$(ABS(A
A(I)))
340 IF ABS(AA(I))<>5 THEN O$(I)=O$(I)&RP
T$(" ",3-LEN(STR$(K)))&STR$(K)
350 GOTO 320
360 DISPLAY ERASE ALL AT(1,1):H$ :: IF (
PF)THEN PRINT #2:SEG$(H$,1,28):SEG$(H$,2
9,56)
370 R=2
380 FOR P=1 TO LF
390 R=R+1
400 DISPLAY AT(R,1):O$(P):: IF (PF)THEN
PRINT #2:O$(P)
410 IF R<>22 THEN 490
420 DISPLAY AT(23,1):STR$(P+1)&" PROGRAM
 ";TAB(13);"(ENTER=CONTINUE)"
```

```
430 ACCEPT AT(23,1)VALIDATE(DEC$&CHR$(13
))BEEP SIZE(-3):@$
440 IF @$=STR$(P+1)THEN R=2 :: CALL HCHA
R(3,1,32,672):: GOTO 490 ELSE @=VAL(@$)
450 IF @<1 OR @>LF THEN M$="MUST BE BETW
EEN 1 AND "&STR$(LF):: GOSUB 690 :: GOTO
 420
460 IF ABS(AA(@))=5 THEN 730
470 IF AA(@)=4 THEN 730
480 M$="MUST BE A PROGRAM FILE !!!" :: G
OSUB 690 :: GOTO 420
490 NEXT P
500 DISPLAY AT(23,1):"NO MORE FILES, SEL
ECT ONE : 000=AGAIN  XXX=PGM  999=STOP"
510 ACCEPT AT(24,1)VALIDATE(DEC$)SIZE(-3
):@$ :: @=VAL(@$)
520 IF @<>0 THEN 550
530 IF (PF)THEN PF=0 :: CLOSE #2
540 GOTO 180
550 IF @<>999 THEN 590
560 IF (PF)THEN CLOSE #2
570 CALL CLEAR :: CALL CHARSET
580 STOP
590 IF @>LF THEN M$="MUST BE BETWEEN 1 A
ND "&STR$(LF):: GOSUB 690 :: GOTO 500
600 IF ABS(AA(@))=5 OR ABS(AA(@))=4 THEN
 730 ELSE M$="MUST BE A PROGRAM FILE !!!
" :: GOSUB 690 :: GOTO 500
610 REM  ACCEPT SUBROUTINE
620 FOR K=1 TO NK :: READ R,C,AR,AC,SZ,V
$,M$ :: DISPLAY AT(R,C):M$ :: ACCEPT AT(
AR,AC)VALIDATE(V$)SIZE(SZ):A$(K):: NEXT
K
630 RETURN
640 DATA 300,DIS/FIX,1200,DIS/VAR,2400,I
NT/FIX,4800,INT/VAR,9600,PROGRAM
650 DATA 12,1,12,24,-1,YNyn,PRINTOUT OF
CATALOG ?  NY
660 DATA 14,1,14,24,-1,12,RS232 PORT NUM
BER    : 12
670 DATA 15,1,15,24,-1,78,NUMBER OF BITS
```

```
 USED  ? 87
680 DATA 16,1,19,18,-1,12345,BAUD RATE (
BITS/SEC)         1=300   2=1200   3=2400
     4=4800            5=9600         YOUR
 CHOICE ? 5
690 FOR I=1 TO 4 :: DISPLAY AT(23,1):BL$
 :: CALL SOUND(-400,110,8):: DISPLAY AT(
23,1):M$ :: CALL SOUND(99,110,30):: NEXT
 I :: RETURN
700 CALL ERR(E,T,S,L)
710 M$="<<< wEiRd0 error "&STR$(E)&" >>>
" :: GOSUB 690
720 RETURN 180
730 REM GENERAL PURPOSE LOADER      99ER
 MAGAZINE                 USED BY PERMISSI
ON
740 CALL CHARSET
750 CALL PEEK(-31952,I,J):: CALL PEEK(I*
256+J-65534,I,J):: K=I*256+J-65534 :: @$
="DSK1."&G$(@):: CALL LOAD(K,LEN(@$))
760 FOR I=1 TO LEN(@$):: CALL LOAD(K+I,A
SC(SEG$(@$,I,1))):: NEXT I :: CALL LOAD(
K+I,0)
770 RUN "DSK1.1234567890"
```

CHRISTMAS BILLBOARD

This is one of the few programs that requires only the basic console. Although this program has limited uses, it is included because it is creative and reveals an extensive use of what console basic can do. You will find the code is somewhat lengthy, but self documenting.

On the left side of the screen the program displays a Christmas tree with wrapped packages underneath. Up to seven messages of your choice are displayed, one at a time, on the right side of the screen. The length of time that each message is displayed is also selectable. The program prompts you for each message and its associated time of display. For however long a message is displayed, the lights on the Christmas tree blink as rapidly as possible; a random number generator is used to select which light to blink.
After the Christmas tree is drawn, the song "I'll be home for Christmas" is played. As each note is played, a light on the Christmas tree blinks.

After the last message has been displayed, the screen is blanked and the program starts over at its beginning, drawing the Christmas Tree and repeating your messages.

When entering a message, as soon as you press ENTER you will see the statement "VALIDATING YOUR MESSAGE" on the screen. If a word is longer than the number of columns allocated for message display, or if the number of words are greater than the dimension of the variable used to hold them, or if the combination of words are such that they will not fit in the amount of space reserved on the screen for message display, an appropriate error message is displayed on the screen. This is probably the most difficult and interesting code in the program.

```
100 REM COPYRIGHT 1983
110 REM JOHN ROBERT COPE
120 REM CRACKING THE 99/4A
130 REM BASIC
140 REM  GET I/P INFO
150 OPTION BASE 1
160 MNO=1
170 CALL CLEAR
180 PRINT "ENTER YOUR CHRISTMAS          M
ESSAGE NO.";MNO;"OF 6": : :
```

```
190 PRINT "   ENCLOSE MESSAGE IN QUOTES
  (HOLD DOWN <FCTN> AND       THEN PRESS
 <P> TO MAKE A    QUOTE).": :
200 PRINT "   EACH MESSAGE CANNOT BE
  LONGER THAN 4 LINES.": :
210 PRINT "   IF YOU WANT A QUOTE IN
  THE MESSAGE, PUT TWO OF     THEM WHERE
 YOU WANT IT.": : :
220 BL$=" "
230 PRINT "IF NO MORE MESSAGES, JUST   P
RESS ENTER.": : :
240 INPUT ">":MSG$(MNO)
250 IF MSG$(MNO)="" THEN 740
260 GOSUB 430
270 CALL CLEAR
280 PRINT "ENTER THE TIME - IN SECONDS -
 THAT YOU WANT THIS MESSAGETO REMAIN ON
THE SCREEN     BEFORE GOING TO THE NEXT
   MESSAGE.": :
290 IF MNO>1 THEN 310
300 PRINT "IF YOU HAVE GIVEN ONLY ONE  M
ESSAGE AND YOU WANT IT TO  BE DISPLAYED
PERMANENTLY,   ENTER ZERO.": :
310 PRINT "ACCEPTABLE VALUES ARE FROM  O
NE (1) TO SIXTY (60)       SECONDS": : :
320 INPUT ">":SEC(MNO)
330 IF MNO>1 THEN 350
340 IF SEC(MNO)=0 THEN 740
350 IF MNO=6 THEN 740
360 IF SEC(MNO)<=60 THEN 380
370 SEC(MNO)=60
380 IF SEC(MNO)>0 THEN 400
390 SEC(MNO)=1
400 MNO=MNO+1
410 GOTO 170
420 REM VALIDATE MSGS
430 CALL CLEAR
440 PRINT " VALIDATING YOUR MESSAGE...":
 : :
450 DIM WD$(6,40),L(13)
460 PTR=1
470 WORD=1
480 SP=POS(MSG$(MNO),BL$,PTR)
490 IF SP<>0 THEN 520
500 WLEN=LEN(MSG$(MNO))-PTR+1
```

```
510 GOTO 570
520 WLEN=SP-PTR
530 IF WLEN<>0 THEN 570
540 REM SIDE BY SIDE BLANKS
550 PTR=PTR+1
560 GOTO 480
570 WD$(MNO,WORD)=SEG$(MSG$(MNO),PTR,WLE
N)
580 IF WLEN>13 THEN 2310
590 REM NO MORE WORDS ?
600 IF SP=0 THEN 670
610 PTR=SP+1
620 WORD=WORD+1
630 IF WORD<=40 THEN 480
640 CALL CLEAR
650 PRINT "SORRY. NO MORE THAN 40       W
ORDS PER SENTENCE."
660 GOTO 3890
670 GOSUB 3720
680 IF SRC=0 THEN 720
690 CALL CLEAR
700 PRINT "SORRY. THIS PARTTICULAR      C
OMBINATION OF WORDS WILL   NOT FIT ON TH
E SCREEN AREA."
710 GOTO 3890
720 RETURN
730 REM ALL VALIDATED
740 CALL CLEAR
750 CALL SCREEN(12)
760 PASS=0
770 LET TRANS=1
780 LET BLACK=2
790 LET MEDGR=3
800 LET LTGR=4
810 LET DKBLU=5
820 LET LTBLU=6
830 LET DKRED=7
840 LET CYAN=8
850 LET MEDRED=9
860 LET LTRED=10
870 LET DKYEL=11
880 LET LTYEL=12
890 LET DKGR=13
900 LET MAGEN=14
910 LET GRAY=15
```

```
920 LET WHITE=16
930 REM DEFINE SHAPES
940 LET CNDL$="FFE7C3C3C3C3C3FF"
950 LET BLOCK$="FFFFFFFFFFFFFFFF"
960 LET EMPTY$="0000000000000000"
970 LET LRTRI$="0103070F1F3F7FFF"
980 LET LLTRI$="80C0E0F0F8FCFEFF"
990 LET ULTRI$="FFFEFCF8F0E0C080"
1000 LET URTRI$="FF7F3F1F0F070301"
1010 LET RBLK$="0F0F0F0F0F0F0F0F"
1020 LET LBLK$="F0F0F0F0F0F0F0F0"
1030 LET TLINE$="FF00000000000000"
1040 LET BLINE$="00000000000000FF"
1050 LET MLINE$="000000FFFF000000"
1060 LET ULST$="0101231109054 3FF"
1070 LET URST$="8080C48890A0C2FF"
1080 LET LLST$="FF43050911210301"
1090 LRST$="FFC2A0908884C080"
1100 REM MAKE THE TREE
1110 CALL CLEAR
1120 CALL CHAR(136,LRTRI$)
1130 CALL CHAR(137,LLTRI$)
1140 CALL COLOR(14,DKGR,LTYEL)
1150 CALL HCHAR(6,9,136)
1160 CALL HCHAR(6,10,137)
1170 CALL CHAR(138,BLOCK$)
1180 CALL HCHAR(7,9,138,2)
1190 CALL HCHAR(8,8,136)
1200 CALL HCHAR(8,9,138,2)
1210 CALL HCHAR(8,11,137)
1220 CALL HCHAR(9,8,138,4)
1230 CALL HCHAR(10,7,136)
1240 CALL HCHAR(10,8,138,4)
1250 CALL HCHAR(10,12,137)
1260 CALL HCHAR(11,7,138,6)
1270 CALL HCHAR(12,6,136)
1280 CALL HCHAR(12,7,138,6)
1290 CALL HCHAR(12,13,137)
1300 CALL HCHAR(13,6,138,8)
1310 CALL HCHAR(14,5,136)
1320 CALL HCHAR(14,6,138,8)
1330 CALL HCHAR(14,14,137)
1340 CALL HCHAR(15,5,138,10)
1350 CALL HCHAR(16,4,136)
1360 CALL HCHAR(16,5,138,10)
```

```
1370 CALL HCHAR(16,15,137)
1380 CALL HCHAR(17,3,136)
1390 CALL HCHAR(17,4,138,12)
1400 CALL HCHAR(17,16,137)
1410 CALL COLOR(13,BLACK,LTYEL)
1420 CALL CHAR(128,RBLK$)
1430 CALL CHAR(129,LBLK$)
1440 CALL HCHAR(18,9,128)
1450 CALL HCHAR(18,10,129)
1460 CALL COLOR(15,GRAY,LTYEL)
1470 CALL CHAR(144,BLOCK$)
1480 CALL CHAR(145,URTRI$)
1490 CALL CHAR(1466,ULTRI$)
1500 CALL HCHAR(19,8,145)
1510 CALL HCHAR(19,11,146)
1520 CALL HCHAR(19,9,144,2)
1530 CALL HCHAR(20,9,144,2)
1540 CALL COLOR(16,LTBLU,LTRED)
1550 CALL CHAR(152,BLOCK$)
1560 CALL CHAR(153,ULTRI$)
1570 CALL CHAR(154,LRTRI$)
1580 CALL HCHAR(19,4,152)
1590 CALL HCHAR(19,5,153)
1600 CALL CHAR(155,EMPTY$)
1610 CALL HCHAR(19,6,155)
1620 CALL HCHAR(20,4,153)
1630 CALL HCHAR(20,5,155)
1640 CALL HCHAR(20,6,154)
1650 CALL HCHAR(21,4,155)
1660 CALL HCHAR(21,5,154)
1670 CALL HCHAR(21,6,152)
1680 CALL COLOR(12,CYAN,DKRED)
1690 CALL CHAR(120,BLOCK$)
1700 CALL CHAR(121,MLINE$)
1710 CALL HCHAR(19,13,120,3)
1720 CALL HCHAR(20,13,121,3)
1730 CALL HCHAR(21,13,120,3)
1740 CALL COLOR(11,WHITE,TRANS)
1750 CALL CHAR(112,ULST$)
1760 CALL CHAR(113,URST$)
1770 CALL CHAR(114,LLST$)
1780 CALL CHAR(115,LRST$)
1790 CALL HCHAR(4,9,112)
1800 CALL HCHAR(4,10,113)
1810 CALL HCHAR(5,9,114)
```

```
1820 CALL HCHAR(5,10,115)
1830 REM DRAW CANDLES
1840 CALL CHAR(139,CNDL$)
1850 CALL CHAR(96,CNDL$)
1860 CALL CHAR(104,CNDL$)
1870 CALL COLOR(9,DKGR,DKBLU)
1880 CALL COLOR(10,DKGR,MEDRED)
1890 CALL HCHAR(10,9,139)
1900 CALL HCHAR(8,10,96)
1910 CALL HCHAR(12,11,104)
1920 CALL HCHAR(13,8,139)
1930 CALL HCHAR(14,6,96)
1940 CALL HCHAR(15,10,104)
1950 CALL HCHAR(14,13,139)
1960 CALL HCHAR(16,8,139)
1970 CALL HCHAR(16,12,96)
1980 CALL HCHAR(17,5,104)
1990 CALL HCHAR(17,14,96)
2000 GOSUB 3190
2010 REM MAINLINE ROUTINE
2020 IF PASS>0 THEN 2060
2030 MNO=1
2040 PASS=1
2050 REM GO WRITE A MSG
2060 GOSUB 2210
2070 IF SEC(MNO)=0 THEN 2110
2080 TWTM=(6*(SEC(MNO)))
2090 IF TWTM=0 THEN 2110
2100 FOR T=1 TO TWTM
2110 GOSUB 2660
2120 IF SEC(MNO)=0 THEN 2110
2130 NEXT T
2140 MNO=MNO+1
2150 IF MNO=8 THEN 2180
2160 IF MSG$(MNO)="" THEN 2180
2170 GOTO 2060
2180 MNO=1
2190 GOTO 1110
2200 REM ERASE
2210 CALL CHAR(140,EMPTY$)
2220 FOR C=18 TO 32
2230 CALL VCHAR(24,C,140,24)
2240 NEXT C
2250 REM INIT
2260 V=3
```

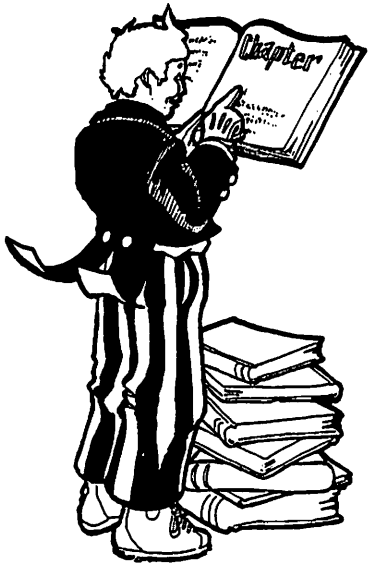
```
2270 WORD=1
2280 H=19
2290 PTR=1
2300 GOTO 2360
2310 CALL CLEAR
2320 PRINT "SORRY. CHARACTER SEQUENCES
CANNOT BE LONGER THAN 13    CHARACTERS."
2330 PRINT
2340 PRINT "THE SEQUENCE ": :"'";WD$(MNO
,WORD);"'": :"IS ";LEN(WD$(MNO,WORD));"
CHARACTERS."
2350 GOTO 3890
2360 FOR NL=1 TO LEN(WD$(MNO,WORD))
2370 L(NL)=ASC(SEG$(WD$(MNO,WORD),PTR,1)
)
2380 PTR=PTR+1
2390 NEXT NL
2400 FOR NL=1 TO LEN(WD$(MNO,WORD))
2410 REM WRITE A WORD
2420 CALL HCHAR(V,H,L(NL))
2430 H=H+1
2440 NEXT NL
2450 WORD=WORD+1
2460 IF WD$(MNO,WORD)<>"" THEN 2480
2470 RETURN
2480 IF H<31 THEN 2560
2490 H=19
2500 V=V+2
2510 IF V<24 THEN 2290
2520 CALL CLEAR
2530 PRINT "SORRY. THIS COMBINATION OF
WORDS WILL NOT FIT ON THE   SCREEN."
2540 IF H<30 THEN 2560
2550 GOTO 2590
2560 CALL HCHAR(V,H,32)
2570 H=H+1
2580 IF LEN(WD$(MNO,WORD))<=32-H THEN 22
90
2590 H=19
2600 V=V+2
2610 IF V<24 THEN 2290
2620 CALL CLEAR
2630 PRINT "SORRY. THIS COMBINATION OF
WORDS WILL NOT FIT ON THE   SCREEN."
2640 GOTO 3890
```

```
2650 REM TWINKLE ONE
2660 RANDOMIZE
2670 STAR=INT((11*RND)+1)
2680 ON STAR GOSUB 2740,2790,2830,2870,2
910,2950,2990,3030,3070,3110,3150
2690 CALL HCHAR(VS,HS,138)
2700 FOR S=1 TO 10
2710 NEXT S
2720 CALL HCHAR(VS,HS,CHAR)
2730 RETURN
2740 VS=10
2750 HS=9
2760 CHAR=139
2770 RETURN
2780 DATA 12,11,104
2790 VS=8
2800 HS=10
2810 CHAR=96
2820 RETURN
2830 VS=12
2840 HS=11
2850 CHAR=104
2860 RETURN
2870 VS=13
2880 HS=8
2890 CHAR=139
2900 RETURN
2910 VS=14
2920 HS=6
2930 CHAR=104
2940 RETURN
2950 VS=15
2960 HS=10
2970 CHAR=104
2980 RETURN
2990 VS=14
3000 HS=13
3010 CHAR=139
3020 RETURN
3030 VS=16
3040 HS=8
3050 CHAR=139
3060 RETURN
3070 VS=16
3080 HS=12
```

```
3090 CHAR=96
3100 RETURN
3110 VS=17
3120 HS=5
3130 CHAR=104
3140 RETURN
3150 VS=17
3160 HS=14
3170 CHAR=96
3180 RETURN
3190 REM MUSIC
3200 QNOTE=375
3210 HNOTE=750
3220 TNOTE=1125
3230 FNOTE=1500
3240 VOL=10
3250 CALL SOUND(TNOTE,262,VOL-5,523,VOL)
3260 GOSUB 2660
3270 CALL SOUND(QNOTE,247,VOL-5,494,VOL)
3280 GOSUB 2660
3290 CALL SOUND(TNOTE,294,VOL-5,587,VOL)
3300 GOSUB 2660
3310 CALL SOUND(QNOTE,262,VOL-5,523,VOL)
3320 GOSUB 2660
3330 CALL SOUND(HNOTE,196,VOL-5,392,VOL)
3340 GOSUB 2660
3350 CALL SOUND(FNOTE,196,VOL-5,392,VOL)
3360 GOSUB 2660
3370 CALL SOUND(TNOTE,220,VOL-5,440,VOL)
3380 GOSUB 2660
3390 CALL SOUND(QNOTE,196,VOL-5,392,VOL)
3400 GOSUB 2660
3410 CALL SOUND(TNOTE,233,VOL-5,466,VOL)
3420 GOSUB 2660
3430 CALL SOUND(QNOTE,220,VOL-5,440,VOL)
3440 GOSUB 2660
3450 CALL SOUND(2*FNOTE,147,VOL-5,294,VO
L)
3460 GOSUB 2660
3470 CALL SOUND(TNOTE,294,VOL-5,587,VOL)
3480 GOSUB 2660
3490 CALL SOUND(QNOTE,262,VOL-5,523,VOL)
3500 GOSUB 2660
3510 CALL SOUND(TNOTE,294,VOL-5,587,VOL)
3520 GOSUB 2660
```

```
3530 CALL SOUND(QNOTE,262,VOL-5,523,VOL)
3540 GOSUB 2660
3550 CALL SOUND(TNOTE,196,VOL-5,392,VOL)
3560 GOSUB 2660
3570 CALL SOUND(TNOTE,220,VOL-5,440,VOL)
3580 GOSUB 2660
3590 CALL SOUND(QNOTE,220,VOL-5,440,VOL)
3600 GOSUB 2660
3610 CALL SOUND(HNOTE,294,VOL-5,587,VOL)
3620 GOSUB 2660
3630 CALL SOUND(HNOTE,330,VOL-5,659,VOL)
3640 GOSUB 2660
3650 CALL SOUND(HNOTE,262,VOL-5,523,VOL)
3660 GOSUB 2660
3670 CALL SOUND(HNOTE,294,VOL-5,587,VOL)
3680 GOSUB 2660
3690 CALL SOUND(2*FNOTE,262,VOL-5,523,VO
L)
3700 GOSUB 2660
3710 RETURN
3720 REM FIT ON PAGE ?
3730 WRD=1
3740 LINE=1
3750 LEFT=13
3760 LEFT=LEFT-LEN(WD$(MNO,WRD))
3770 IF LEFT<0 THEN 3820
3780 WRD=WRD+1
3790 IF WRD>40 THEN 3870
3800 IF WD$(MNO,WRD)="" THEN 3870
3810 GOTO 3760
3820 LINE=LINE+1
3830 IF LINE>11 THEN 3850
3840 GOTO 3750
3850 SRC=4
3860 RETURN
3870 SRC=0
3880 GOTO 3860
3890 PRINT
3900 PRINT "PRESS ENTER TO RE-ENTER
MESSAGE";MNO;"."
3910 CALL KEY(0,RV,SV)
3920 IF SV=0 THEN 3910
3930 FOR I=1 TO 40
3940 WD$(MNO,I)=""
3950 NEXT I
3960 GOTO 170
```

SPEECH



Speech unit
TE II cartridge

SPEECH CONTROL

The TI SOLID STATE SPEECH SYNTHESIZER is a small add on peripheral for the TI99/4A HOME COMPUTER. The speech unit measures less than 2 1/4 inches wide, and 5 inches deep. The unit plugs into the right hand side of the Computer Console. The Peripheral Expansion cable attaches to the right hand side of the Speech Module, so that the Speech Module fits between the Console and the Expansion Box.

Addition of speech brings spice to programs that are dull and lifeless. To utilize speech in a program requires the use of a command module that provides grom routines that access the speech unit, or use of Extended Basic, Assembly, or UCSD PASCAL programing languages.

USE WITH OTHER COMMAND MODULES

The easiest way to utilize speech within a program is to use Ti Basic with a command module like the Speech Editor, or the TEII Terminal Emulator Command cartridge. Each of these modules provide Speech firmware support via their groms. In most cases, the user programs speech I/O in the same manner as that of a variable or a fixed sequential file.

The following programs illustrates this point. Notice that output to the speech unit is also similar to that of output to the crt screen.

```
100 REM  SPEECH EXAMPLE
110 REM
120 REM  PURPOSE: TO ALLOW EXPERIMENTATI
ON OF SPEECH PHRASES
130 REM  WITH THE TEII COMMAND MODULE
140 REM
150 REM  REQUIRES USE OF TEII COMMAND MO
DULE
160 REM
170 REM  WRITTEN BY JOEL RODRIGUEZ 12/83
180 REM   ALL RIGHTS ARE RESERVED BY AUT
HOR
```

```
190 REM
200 REM    ****     PROGRAM START     ****
210 REM    INIT VARS,SCREEN,& SPEECH UNIT
220 REM
230 LET DEFAULT$="//43 128"
240 LET UDEF$="//43 128"
250 LET T$="WHAT DO YOU WANT TO SAY"
260 CALL CLEAR
270 OPEN #1:"SPEECH",OUTPUT
280 PRINT #1:DEFAULT$
290 REM
300 REM  OUTPUT MAIN TITLE SCREEN
310 REM
320 GOSUB 670
330 REM
340 REM     RUN DEFAULT MODE(?)
350 REM
360 GOSUB 1340
370 IF D$="Y" THEN 420
380 GOSUB 1480
390 REM
400 REM  GET SPEECH TEXT
410 REM
420 GOSUB 1210
430 REM
440 REM  EXECUTE SPEECH
450 REM
460 GOSUB 1800
470 GOSUB 2460
480 IF D$="Y" THEN 510
490 GOSUB 1920
500 GOTO 520
510 PRINT #1:T$
520 GOSUB 2460
530 REM
540 REM  END PROGRAM ACTION
550 REM
560 GOSUB 2170
570 IF A$="Y" THEN 470
580 IF A$="C" THEN 420
590 IF A$="M" THEN 360
600 END
610 REM
620 REM  SUBROUTINES ***************
630 REM
```

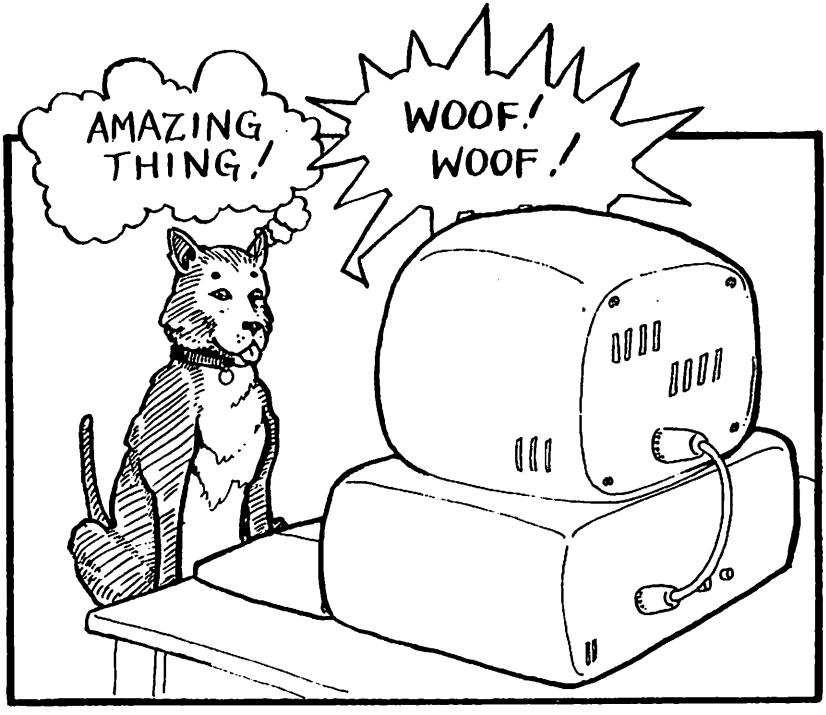
```
640 REM
650 REM  INSTRUCTIONS
660 REM
670 CALL SCREEN(4)
680 PRINT "SPEECH PITCH & SLOPE TESTER"
690 PRINT #1:"SPEECH PITCH AND _SLOPE _T
ESTER"
700 PRINT "  "
710 PRINT "DEFAULT MODE ALLOWS REPEATED
EXECUTION OF THE USER SET"
720 PRINT " PITCH & SLOPE PARAMETERS."
730 PRINT "  "
740 PRINT "NON-DEFAULT MODE ALLOWS RE-"
750 PRINT " PEATED EXECUTION OF VAR-"
760 PRINT " IABLE PITCH & SLOPE PARMS."
770 PRINT "  "
780 FOR DELAY=1 TO 500
790 NEXT DELAY
800 RETURN
810 REM
820 REM  CHANGE DEFAULT
830 REM
840 PRINT "  "
850 PRINT "THE DEFAULT PROGRAM PITCH &
SLOPE PARMS ARE ://43 128."
860 PRINT "  "
870 PRINT "CURRENT USER DEFAULT PARMS
ARE :"&UDEF$&"."
880 PRINT "  "
890 IF E$<>"Y" THEN 920
900 PRINT #1:"CHANGE DEFAULT"
910 GOSUB 2520
920 INPUT "CHANGE DEFAULT(Y/N) - ":A$
930 IF A$="Y" THEN 950
940 GOTO 1070
950 IF E$<>"Y" THEN 970
960 PRINT #1:" ENTER NEW DEFAULT"
970 INPUT "ENTER NEW DEFAULT(//XX YYY) :
 ":UDEF$
980 PV$=SEG$(UDEF$,3,2)
990 SV$=SEG$(UDEF$,6,3)
1000 IF PV$<" 0" THEN 1050
1010 IF PV$>"63" THEN 1050
1020 IF SV$<"  0" THEN 1050
1030 IF SV$>"255" THEN 1050
```

```
1040 GOTO 1070
1050 GOSUB 1110
1060 GOTO 960
1070 RETURN
1080 REM
1090 REM  OUTPUT VALID PARMS
1100 REM
1110 PRINT "  "
1120 IF E$<>"Y" THEN 1140
1130 PRINT #1:"INVALID"
1140 PRINT "XX IS PITCH, RANGE 0-63."
1150 PRINT "YYY IS SLOPE, RANGE 0-255."
1160 PRINT "  "
1170 RETURN
1180 REM
1190 REM  INPUT SPEECH TEXT
1200 REM
1210 PRINT "  "
1220 PRINT "CURRENT SPEECH TEXT IS :"
1230 PRINT "  "
1240 PRINT T$
1250 PRINT "  "
1260 IF E$<>"Y" THEN 1280
1270 PRINT #1:"ENTER SPEECH TEXT"
1280 PRINT "ENTER SPEECH TEXT"
1290 INPUT T$
1300 RETURN
1310 REM
1320 REM  RUN DEFAULT MODE
1330 REM
1340 CALL SCREEN(8)
1350 PRINT "  "
1360 INPUT "ECHO PROMPTS TO SPEECH UNIT
(Y/N) - ":E$
1370 PRINT "  "
1380 IF E$<>"Y" THEN 1410
1390 PRINT #1:"DEFAULT MODE"
1400 GOSUB 2520
1410 INPUT "DEFAULT MODE(Y/N) - ":D$
1420 IF D$="N" THEN 1440
1430 GOSUB 840
1440 RETURN
1450 REM
1460 REM  PROMPT VARIABLE PARMS
1470 REM
```

```
1480 PRINT "  "
1490 IF E$<>"Y" THEN 1510
1500 PRINT #1:"ENTER PITCH RANGE"
1510 PRINT "ENTER PITCH RANGE"
1520 PRINT "  "
1530 IF E$<>"Y" THEN 1550
1540 PRINT #1:"STARTING VALUE IS"
1550 INPUT "STARTING VALUE(0-63) : ":PS
1560 IF E$<>"Y" THEN 1580
1570 PRINT #1:"ENDING VALUE IS"
1580 INPUT "ENDING VALUE(0-63) : ":PE
1590 IF E$<>"Y" THEN 1610
1600 PRINT #1:"INCREMENT BY"
1610 INPUT "INCREMENT VALUE : ":PI
1620 PRINT "  "
1630 IF E$<>"Y" THEN 1650
1640 PRINT #1:"ENTER SLOPE RANGE"
1650 PRINT "ENTER SLOPE RANGE"
1660 PRINT "  "
1670 IF E$<>"Y" THEN 1690
1680 PRINT #1:"STARTING VALUE IS"
1690 INPUT "STARTING VALUE(0-255) : ":SS
1700 IF E$<>"Y" THEN 1720
1710 PRINT #1:"ENDING VALUE IS"
1720 INPUT "ENDING VALUE(0,255) : ":SE
1730 IF E$<>"Y" THEN 1750
1740 PRINT #1:"INCREMENT VALUE BY"
1750 INPUT "INCREMENT VALUE : ":SI
1760 RETURN
1770 REM
1780 REM   OUTPUT START
1790 REM
1800 CALL SCREEN(12)
1810 PRINT "  "
1820 PRINT "STARTING SPEECH TEST RUN"
1830 IF E$<>"Y" THEN 1850
1840 PRINT #1:"STARTING SPEECH TEST RUN"
1850 PRINT #1:UDEF$
1860 PRINT "  "
1870 GOSUB 2460
1880 RETURN
1890 REM
1900 REM  EXECUTE PARMS
1910 REM
1920 FOR PITCH=PS TO PE STEP PI
```

```
1930 FOR SLOPE=SS TO SE STEP SI
1940 LET P$=""
1950 IF PITCH>9 THEN 1970
1960 P$="0"
1970 P$=P$&STR$(PITCH)&" "
1980 LET S$=""
1990 IF SLOPE>99 THEN 2010
2000 S$=S$&"0"
2010 IF SLOPE>9 THEN 2030
2020 S$=S$&"0"
2030 S$=S$&STR$(SLOPE)
2040 C$="//"&P$&S$
2050 PRINT #1:C$
2060 PRINT "  "
2070 PRINT "PITCH ="&P$&" SLOPE ="&S$,"
CONTROL ="&C$
2080 PRINT #1:T$
2090 CALL KEY(0,K,S)
2100 IF S<>0 THEN 2130
2110 NEXT SLOPE
2120 NEXT PITCH
2130 RETURN
2140 REM
2150 REM  END ACTION
2160 REM
2170 CALL SCREEN(4)
2180 PRINT "  "
2190 IF E$<>"Y" THEN 2230
2200 PRINT #1:DEFAULT$
2210 PRINT #1:"RUN ^AGAIN"
2220 GOSUB 2520
2230 INPUT "RUN AGAIN(Y/N) - ":A$
2240 IF A$="Y" THEN 2420
2250 PRINT "  "
2260 IF E$<>"Y" THEN 2290
2270 PRINT #1:"CHANGE SPEECH TEXT"
2280 GOSUB 2520
2290 INPUT "CHANGE SPEECH TEXT(Y/N) - ":
A$
2300 IF A$<>"Y" THEN 2330
2310 A$="C"
2320 GOTO 2420
2330 PRINT "  "
2340 IF E$<>"Y" THEN 2370
2350 PRINT #1:"CHANGE MODE"
```

```
2360 GOSUB 2520
2370 INPUT "CHANGE MODE(Y/N) - ":A$
2380 IF A$<>"Y" THEN 2410
2390 A$="M"
2400 GOTO 2420
2410 STOP
2420 RETURN
2430 REM
2440 REM  DELAY ROUTINE
2450 REM
2460 FOR DELAY=1 TO 250
2470 NEXT DELAY
2480 RETURN
2490 REM
2500 REM  MISC SPEECH PROMPTS
2510 REM
2520 PRINT #1:"YES. OR. NO"
2530 RETURN
2540 END
```



The program was conceived to allow experimental output to the speech unit, and also to provide program development capabilites for the speech programmer. First a brief description of the code is in order.

Lines 100-320 : Program start text, and program initialization code.

Lines 330-600 : Code to select mode and parameters, input speech text, speek text, and end program action

Lines 640-800 : TITLE SCREEN MENU

Lines 810-1070 : Code to set execution mode

Lines 1080-1300 : Messages

Lines 1310-1760 : Mode prompts

Lines 1770-1880 : Starting Execution message

Lines 1890-2130 : Variable execution control

Lines 2140-2420 : End action prompts

Lines 2430-2530 : Misc subroutines

USING THE PROGRAM:

As well as experimenting with speech itself, program may be used for determining which pitch and slope parameters are required to effect the voice quality of the resulting speech. The tonal quality may be varied from hi to lo, or set at a whisper. The lower pitch values produce a hi mouse-like voice while the higher pitch values produce a deep masculine voice. The slope may be varied to some degree to shape the resulting speech. The following two equations govern the recommended slope value for a particular pitch, and a range of of valid slopes for a particular pitch. Exceeding the valid range of slope will garble the speech; with some interesting and some not so interesting results.

RECOMMENDED : SLOPE = 32 * INT(.1 * PITCH)

RANGE : SLOPE < (PITCH—1) * 16 or
SLOPE < (63—PITCH) * 16

The program may also be used to vary the stress points of a sentence, and to add pauses and delays. A sentence may have it's primary stress point word preceded by the ± symbol. Secondary stress points may be indicated with the __ symbol preceding the word. Note that a sentence may have only one primary stress point, but can have multiple secondary stress points. To use these inflection symbols just type them in preceding the word in the sentence where you would like to change the stress points.

Pauses may be implemeted with the '. ' or ', ' characters appended to the appropriate word. The '. ' sequence will cause a .45 second delay while the ', ' sequence will cause a .1 second delay after the word is spoken. Each of these symbols consist of two characters. The comma or period followed by a space.

In addition, the contour of the word may be altered to either rise or fall. The contour refers only to the stress point of the sentence and may only be used on words that are preceded by an inflection symbol.(± or __) The '. ' and ? symbols both specify a rising contour, while the symbols ', ', !, :, and ; all specify falling contours! CONFUSED? YOU BET!

So what does all this mean to you? Well, back to the program. Use the program to experiment. Type in a sentence and listen. Change the sentence. Listen again. Soon you will be able to determine what works and what doesn't work.

TO USE THE PROGRAM:

1] Insert the TEII Command Module and bring up Ti Basic.

2] Load the program.

3] Run the program.

4] The program will clear the screen and if the speech unit is attached, will speak the title of the program.

The Title menu appears as follows:

"SPEECH PITCH & SLOPE TESTER"

"DEFAULT MODE ALLOWS REPEATED"
"EXECUTION OF THE USER SET PARAMTERS"

"NON-DEFAULT MODE ALLOWS RE-"
"PEATED EXECUTION OF VAR-"
"IABLE PITCH & SLOPE PARMS."

5] The program will prompt for initial values and desired mode of operation, as follows :

"ECHO PROMPTS TO SPEECH UNIT (Y/N)—"

If you want the program to speak the prompts then answer Y, else prompts appear on the computer monitor only.

"DEFAULT MODE(Y/N)—"
If you wish to use the default mode of the program, then enter Y, else enter a N to change the program mode of operation.

If you choose the default mode the following sequence of prompts appears:

"THE DEFAULT PROGRAM PITCH & SLOPE PARMS ARE ://43 128
"CURRENT USER DEFAULT PARMS ARE : //XX YYY"

"CHANGE DEFAULT(Y/N)—"

If you wish to set the default pitch and slope parameters, then enter Y, else enter N to use the default, or last setting entered.

If you enter a Y, then the following prompt appears:

"ENTER NEW DEFAULT(//XX YYY) : "

Enter your pitch and slope setting. If you incorrectly enter the values, the program will will respond with the following messages:

"XX IS PITCH, RANGE Ø-63."
"YYY IS SLOPE, RANGE Ø-255."

You are then reprompted for the user default pitch and slope parameters.

If you entered N to the default mode prompt, then the following sequence of prompts appears:

"ENTER PITCH RANGE"
"STARTING VALUE(Ø-63) : "
"ENDING VALUE(Ø-63) : "
"INCREMENT VALUE : "

"ENTER SLOPE RANGE"
"STARTING VALUE(Ø-255) : "
"ENDING VALUE(Ø,255) : "
"INCREMENT VALUE : "

The program then displays the last speech text data input.

"CURRENT SPEECH TEXT IS :"

"ENTER SPEECH TEXT"

Enter the text as desired and press the enter key. The program will respond by displaying the following prompt:

"STARTING SPEECH TEST RUN"

6 The program executes.

7] The program prompts whether or not to continue in the current mode. Answer Y to reexecute the program in the current mode. Answering anything else results in the opportunity to change the mode of operation or exit the program.

Speech unit
TE II cartridge

PHONETIC SPEECH EDITOR

Are you trying to get your speech unit to pronounce words that are understandable, while misspelling and patching your vocabulary together? Tired of doing endless searches for that particular allophone that would solve your pronunciation problems? Here is a program to help you along in your struggle with the Terminal Emulator II cartridge.

The speech utility listed below allows a programmer to exchange allophones in a given phrase in order to achieve proper pronunciation. Phrases can be made phonetically correct without having to misspell any words. Once a proper set of allophones are selected they can be incorporated into your programs.

You may enter up to one line of text at a time, and can do three operations on the phrase.

1. SWAP > Choose and swap allophone from the alternates list.
2. NEW > Clear and enter a new phrase.
3. SPEAK> Speak customized phrase.

The phrase you enter appears vertically on the screen with the corresponding allophone numbers beside each letter. The cursor can be moved up and down the phrase until the desired letter is chosen. At this point pressing the SWAP command will list all the words with similiar sounding allophones. The chosen letter will appear capitalized within each word.

Example: The letter S' as in Ship, or S' as in talkS.

The cursor may now moved down the alternates list until an appropriate substitution letter for the phrase is found. Pressing <ENTER> then places the chosen allophone into your origional phrase. The phrase now may be spoken with the SPEAK command. When the phrase is correct, copy down the allophone numbers next to the phrase for use in your programs.

The following shows how to use the correct allophone numbers in your program.

Allophone numbers for the word HAPPY are;

119,26,1Ø9,25

You would: OPEN #1:"ALPHON",INTERNAL

Then: PRINT #1:CHR$(119)&CHR$(26)&CHR$(1Ø9)&CHR$(25)

Or
Set: A$=CHR$(119)&CHR$(26)&CHR$(1Ø9)&CHR$(25)

Then: PRINT #1:A$

Note pages 37 through 4Ø in the booklet that comes with the TEII module. Especially, the second program on page 39. This sections explains the use of additional numbers at the beginning of your allophone numbers that effect the final sound of your phrase. Also see page 2Ø7 of appendix M in the Extended Basic manual, line numbers 53Ø on down, for an example on how to speak a given allophone string.

```
1Ø REM CRACKING THE 99/4A
2Ø ALLOPHONE EDITOR
3Ø BRUCE WYCHE Ø2/Ø8/84
1ØØ OPTION BASE 1
11Ø DIM A$(125),ALO(125),B(22),M(26),O(2
6)
12Ø CALL CLEAR
13Ø CALL SCREEN(2)
14Ø TRUE=-1
15Ø OPEN #1:"SPEECH",OUTPUT
16Ø OPEN #2:"ALPHON",INTERNAL
17Ø SPEAK=TRUE
18Ø M$="TYPE PHRASE,                  PRES
S ENTER                  1    2    3
            NEW SPEAK SWAP"
19Ø PRINT "  PHONETIC  SPEECH  EDITOR"::
::::::::::::::::
2ØØ PRINT "   LOADING..PLEASE WAIT.."
21Ø CALL SCREEN(15)
22Ø RESTORE
23Ø FOR I=1 TO 121 STEP 5
24Ø READ A$(I),A$(I+1),A$(I+2),A$(I+3),A
$(I+4)
25Ø NEXT I
26Ø O(1)=Ø
27Ø J=Ø
28Ø RESTORE 192Ø
29Ø FOR LETTER=1 TO 26
3ØØ READ M(LETTER)
```

```
310 FOR I=1 TO M(LETTER)
320 J=J+1
330 READ ALO(J)
340 NEXT I
350 IF LETTER=1 THEN 370
360 O(LETTER)=O(LETTER-1)+M(LETTER-1)
370 NEXT LETTER
380 REM   !!! MAIN LOOP !!!
390 GOSUB 1090
400 GOSUB 540
410 IF ((KK<49)+(KK>51))<0 THEN 400
420 ON KK-48 GOSUB 1110,1650,1180
430 GOTO 400
440 REM   DISPLAY M$ @(R,C)
450 C=CL
460 FOR CC=1 TO LEN(M$)
470 CALL HCHAR(RW,C+CC-1,ASC(SEG$(M$,CC,
1)))
480 NEXT CC
490 IF SPEAK<>TRUE THEN 520
500 PRINT #1:M$
510 SPEAK=0
520 RETURN
530 REM  KEY SCAN W/CURSOR
540 CALL KEY(5,KK,SS)
550 CALL HCHAR(ROW,COL,30)
560 CALL HCHAR(ROW,COL,32)
570 IF SS=0 THEN 540
580 CALL SOUND(90,1000,1)
590 RETURN
600 REM  GET PHRASE SUBR
610 X$=""
620 PRINT #1:"ENTER FRAYZ. PLEEZ."
630 CNT=1
640 CALL GCHAR(ROW,COL,GC)
650 CALL KEY(5,KK,SS)
660 CALL HCHAR(ROW,COL,30)
670 CALL HCHAR(ROW,COL,GC)
680 CNT=CNT+1
690 IF CNT>250 THEN 620
700 IF SS=0 THEN 650
710 IF KK>31 THEN 760
720 IF KK=13 THEN 800
730 COL=COL-(COL>2)*(KK=8)+(KK=9)*(COL<3
2)
```

```
740 X$=SEG$(X$,1,COL)
750 GOTO 650
760 CALL HCHAR(ROW,COL,KK)
770 X$=X$&CHR$(KK)
780 COL=COL-(COL<32)
790 GOTO 650
800 CALL HCHAR(1,1,32,704)
810 PRINT #1:X$
820 INPUT #2:N$
830 NL=LEN(N$)
840 IF (NL-3)<23 THEN 870
850 NL=22
860 N$=SEG$(N$,1,NL)
870 J=1
880 FOR I=4 TO NL
890 V=ASC(SEG$(N$,I,1))
900 B(J)=V
910 J=J+1
920 M$=STR$(V)
930 RW=I-3
940 CL=3
950 GOSUB 440
960 M$=A$(V)
970 CL=7
980 GOSUB 440
990 NEXT I
1000 RETURN
1010 M$="UP  DOWN  ENTER"
1020 RW=21
1030 CL=2
1040 GOSUB 440
1050 RETURN
1060 M$="....WORKING...."
1070 GOSUB 440
1080 RETURN
1090 CALL CLEAR
1100 PRINT M$;
1110 CALL HCHAR(1,1,32,640)
1120 SPEAK=TRUE
1130 ROW=20
1140 COL=2
1150 REM  GET PHRASE
1160 GOSUB 610
1170 RETURN
1180 ROW=1
```

```
1190 COL=2
1200 GOSUB 1010
1210 GOSUB 540
1220 IF KK=13 THEN 1250
1230 ROW=ROW-(ROW>1)*(KK=11)+(ROW<(NL-3)
)*(KK=10)
1240 GOTO 1210
1250 GOSUB 1060
1260 REM  GET LETTER FROM ALO
1270 FF=0
1280 FOR JJ=1 TO LEN(A$(B(ROW)))
1290 Z$=SEG$(A$(B(ROW)),JJ,1)
1300 IF Z$<="Z" THEN 1320
1310 NEXT JJ
1320 REM  PRINT ALO-MATCHES
1330 IF Z$=OS$ THEN 1480
1340 LETTER=ASC(Z$)-64
1350 FOR I=1 TO 21
1360 CALL HCHAR(I,18,32,14)
1370 NEXT I
1380 FOR RW=1 TO M(LETTER)
1390 ZZ=ALO(O(LETTER)+RW)
1400 M$=STR$(ZZ)
1410 CL=18
1420 GOSUB 440
1430 M$=A$(ZZ)
1440 CL=22
1450 GOSUB 440
1460 NEXT RW
1470 OS$=Z$
1480 N=ROW
1490 GOSUB 1010
1500 ROW=1
1510 COL=17
1520 GOSUB 1720
1530 B(N)=ZZ
1540 M$=STR$(ZZ)
1550 RW=N
1560 CL=3
1570 CALL HCHAR(RW,CL,32,14)
1580 GOSUB 440
1590 M$=A$(ZZ)
1600 CL=7
1610 GOSUB 440
1620 ROW=N
```

```
163Ø COL=2
164Ø RETURN
165Ø REM      SPEAK PHRASE
166Ø B$=CHR$(25Ø)&CHR$(255)&CHR$(NL-3)
167Ø FOR I=1 TO NL-3
168Ø B$=B$&CHR$(B(I))
169Ø NEXT I
17ØØ PRINT #2:B$
171Ø RETURN
172Ø REM      SUB SWAP ALO
173Ø GOSUB 54Ø
174Ø IF KK=13 THEN 177Ø
175Ø ROW=ROW-(ROW>1)*(KK=11)+(ROW<M(LETT
ER))*(KK=1Ø)
176Ø GOTO 173Ø
177Ø ZZ=ALO(O(LETTER)+ROW)
178Ø RETURN
179Ø DATA Addition,Annuity,deltA
18ØØ DATA ON,AUtonomy,anOnimiity
181Ø DATA Eliminate,Enough,contEXt,ancIE
nt,westERn,synthEsIs
182Ø DATA Inane,tOOk,dOnation,annUal,Uni
que,Above,instrUment,Underneath
183Ø DATA rosEs,basemEnt,seekEr,ratiO,fu
nnY,hAt,hOt,hEIght,cARt,hOUse,sOUght
184Ø DATA hEAt,pIERce,sEt,thERapy,tAke,h
Urt,Issue,chOIce,cOOk,pOORly
185Ø DATA hORse,bOAt,shOOt,hUt,bOOt,hAd,
Odd,hIde
186Ø DATA cARd,lOUd,sAw,sEEd,hEEl,hEAR,s
AId,thERE,dAY,hEARd,hId,hIll,thINk,bOY
187Ø DATA cOUld,pOOR,cORE,lOW,shOE,mUd,s
kULL,pULL,mOOn,Like,bowL,weLL,May
188Ø DATA huM,Nice,saNe,thiNk,thiNG,Real
,Witch,WHich,You,Bad,daB,Dig,biD,Give
189Ø DATA Go,baG,Jug,buDGE,THis,cloTHE,V
ine,aliVE,Zoo,doeS,aZure,beiGE,sKate,Cas
e
19ØØ DATA maKe,Key,Cough,sPace,Pie,naP,s
Take,Tie,laTe,CHurch,Fat,lauGH,Hit,Home
191Ø DATA Hut,Seem,miSS,SHine,waSH,THing
,wiTH
192Ø DATA 9,1,3,18,26,36,47,52,56,58
193Ø DATA 2,86,87
194Ø DATA 3,1Ø4,1Ø7,114
```

```
195Ø DATA 2,88,89
196Ø DATA 18,7,8,9,1Ø,11,12,21,22,23,32,
33,34,35,53,54,55,57,59
197Ø DATA 1,115
198Ø DATA 5,9Ø,91,92,94,1Ø2
199Ø DATA 4,116,117,118,119
2ØØØ DATA 6,13,28,38,49,6Ø,61
2Ø1Ø DATA 1,93
2Ø2Ø DATA 3,1Ø3,1Ø5,1Ø6
2Ø3Ø DATA 4,7Ø,73,74,75
2Ø4Ø DATA 2,76,77
2Ø5Ø DATA 5,62,78,79,8Ø,81
2Ø6Ø DATA 24,4,6,14,15,24,27,3Ø,31,39,4Ø
,41,42,43,44,46,48,51,63,64,65,66,67,68,
72
2Ø7Ø DATA 3,1Ø8,1Ø9,11Ø
2Ø8Ø DATA Ø
2Ø9Ø DATA 3,29,5Ø,82
21ØØ DATA 5,1ØØ,12Ø,121,122,123
211Ø DATA 7,95,96,111,112,113,124,125
212Ø DATA 9,5,16,17,19,2Ø,37,45,69,71
213Ø DATA 2,97,98
214Ø DATA 2,83,84
215Ø DATA Ø
216Ø DATA 3,2,25,85
217Ø DATA 2,99,1Ø1
```

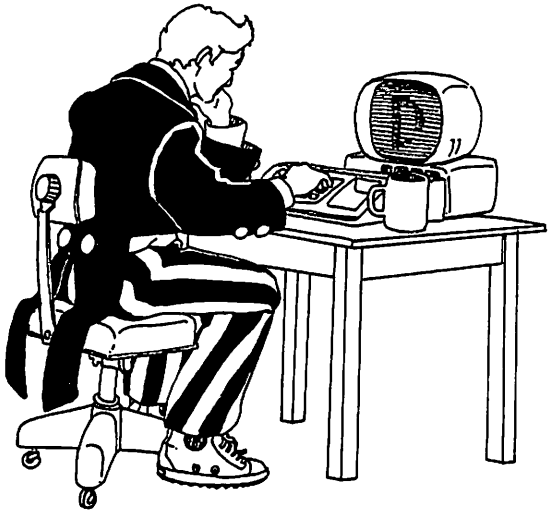
UTILITY

GREAT GRAPHICS GENERATOR

Although computers leave a lot to be desired in lending creative talent, they can help make your job a lot easier. This program allows you to create graphic images dot by dot (up to 32x32 pixels) and then prints out a hexidecimal code equivalent to use for your program graphics. From here the computer asks a few questions, invokes a few magic words, and presto! The image appears actual size, in color, and with motion. When the images are drawn to form, the code then may be saved for future reference or copied into your programs.

The computer first asks which of four available windows you would like to fill in. Keys one through eight move the cursor around a MAGNIFIED version of a 16x16 pixel area. In effect you are redefining four 8 x 8 pixel characters in one window. Having positioned the cursor over the desired position, keys nine and zero turn the dot on or off. This way you can experiment as you go or can copy artwork from a predrawn image from graph paper.

I suggest you try the program as you read the description.



Since there are four windows that may be drawn and put together, you have the capability of redefining 16 characters, drawing a 32 x 32 pixel area. One window is equivalent to one sprite. In effect this program puts four sprites together in order to display all four windows.

From the menu you may choose to:

1. DRAW a chosen window.
2. SEE all windows displayed.
3. Display hex CODES for all windows.
4. EDIT a chosen window.

DRAW

When a window to draw is selected, the computer clears that window's memory. Be careful not to select a window that has already been defined and has not been saved. A small display on the right will show the numbers of all USED windows that have been previously defined. Selecting a NEW window and pressing ENTER immediately thereafter clears the chosen windows memory and deletes it from the USED WINDOW display.

Keys one through eight move the cursor around the window while keys nine and zero turn the dot or pixel on and off. When finished, press ENTER and the computer will calculate and display the hex code equivalent of the design. The four hexidecimal strings that will appear on the screen correspond to the image in the same order that hex codes correspond to sprites.

Char. #1—Upper left corner
Char. #2—Lower left corner
Char. #3—Upper right corner
Char. #4—Lower right corner

Also, like the four characters that make up one window, the four windows also fit together in the same order. When coding a composite image using two or more windows, the window numbers must be assigned to your image corresponding to the order shown below so that the pieces of your composite image will appear on the screen in their proper positions. This is the order in which the program displays windows on the screen, though your own programs may display graphics in any way desired.

Sprite #1. Upper left corner. (Window 1)
Sprite #2. Lower left corner. (Window 2)
Sprite #3. Upper right corner. (Window 3)
Sprite #4. Lower right corner. (Window 4)

EDIT

You may EDIT any window. The computer will redisplay any previously drawn window on the screen so that it may be modified without redrawing

the entire image. After modifying the window, ENTER is pressed to recalculate the new hex code.

SEE

Selecting the SEE option displays all four windows in one composite image. When the screen appears, you are prompted to enter the magnification, X and Y values for motion, sprite image color and screen color (use numeric values). All colors except transparent are available. Calling a black screen will change the color of the characters so that the information on the screen is readable. All parameters may be changed as many times as desired in order to duplicate the appearance of the images for a program.

CODE

When the CODE option is selected, all hex codes are displayed and the word AVAILABLE is displayed for each empty window. Remember to copy down any codes before you clear or alter a window.

TIPS

There are several options for displaying graphics. Motionless graphics may be displayed by redefining the existing character sets using CALL CHAR. The CALL HCHAR or CALL VCHAR statement may be used to display the redefined characters. You can then piece the image together by placing the redefined characters onto the screen in the desired rows and columns. If you are going to have printed messages on the screen with your graphics, remember not to redefine the upper or lowercase alphabet set to be used.

The second option displays a larger image using the CALL SPRITE statement. This allows you to display an image occupying a 16x16 pixel area. The CALL SPRITE statement lets you manipulate the motion, color and position of the sprite in one statement. One window is the largest image a sprite may display (16x16 pixels).

CODE DESCRIPTION

The CALL LOAD(-318Ø6,16) in line 13Ø may be new to you. This disables the quit key so that you cannot quit the program accidently.

The information for a window, when input by a user, is stored in the three dimensional array G(X,R,C), where X equals the number of a window (one through four), and R and C are each 16 elements corresponding to the 16 rows and columns in a window (same as pixels in a sprite).

The movement of the cursor around a window in the program reflects the movement around the 16 X 16 elements of the array. If a dot is turned on in the window, a value of "1" is stored in that position of the array. If a dot is turned off a value of "Ø" is stored. It is also from these ones and zeros that the edit function of the program redisplays a previously drawn window. It scans the array row by row displaying a corresponding dot on

the screen for each value of one it encounters in the array.

Lines 240-310 display the menu command line. Lines 320-350 allow you to choose a window. Lines 410-450 flash the cursor and display a local dot whether it is on or off, and Lines 480-570 are the code for moving the cursor around the screen in the eight available directions.

Lines 590-690 create the cursor wrap around feature. This cuts time when moving from one side of the window to the other. Lines 620 and 630, although somewhat misplaced, are the code which assign a value of one or zero to the array G(X,R,C).

Lines 710-790 keep track of and display which windows are used by looking at the value of A$(I) which contains the hex code equivalent for the corresponding windows (64 characters long). The RPT$ code in line 760 checks to see if the string variable A$(I) is greater than the string value of RPT$("0",64), a string of 64 zeros. If A$(I) is greater than RPT$("0",64) then the window has been previously used and contains hex code.

Lines 1490-1580 loop through all the positions of the array and create the dual function of clearing the values in a given array, or of redisplaying a window for the edit function depending on the command chosen. Under the NEW window option line 1500 checks to see if A$(X)<="0" this says that the chosen window is already empty and causes the clear window code (which fills the array with zeros) to be skipped, thus saving time. This code is also skipped if QQ$="E". This says you are in the EDIT mode and enables line 1540 to reprint an image from the predrawn windows array.

Row by row, the code in lines 1640-1710 convert the 256 elements in the array G(X,R,C) to a windows hexidecimal equivalent. A string 64 characters long. It is this converted code which is used by the CALL SPRITE statements in lines 1190-1220 to display the window under the SEE option.

```
100 REM GRAPHICS GENERATOR
110 REM CRACKING THE 99/4A
120 CALL INIT
130 CALL LOAD(-31806,16)
140 CALL LOAD(-31878,4)
150 DIM G(4,16,16),C$(16)
160 CALL CLEAR :: CALL SCREEN(6):: GOSUB
 810
170 CALL COLOR(12,5,16,9,6,16,10,2,16)
180 CALL CHAR(124,"0303030303030303FF")
190 CALL CHAR(125,"FFFF8080808080FF")
200 CALL CHAR(101,"8080808080808080FF")
210 CALL CHAR(104,"FF8181818181FF")
220 CALL CHAR(105,"FFFFFFFFFFFFFFFF")
```

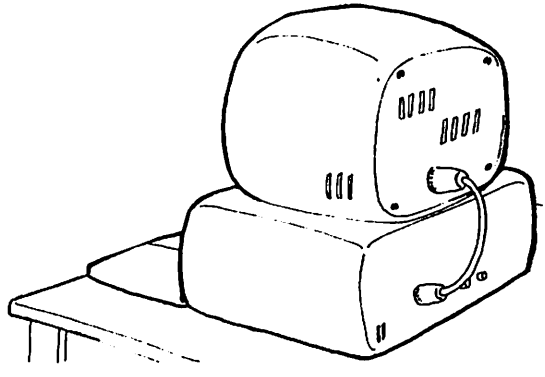
```
230 REM MENU
240 DISPLAY AT(7,10):"MENU":,:"N=DRAW WI
NDOW":,:"E=EDIT OLD WINDOW":,:"S=SEE WIN
DOWS":,:"C=SEE WINDOW CODES"
250 DISPLAY AT(23,1):"N=NEW  S=SEE  C=CO
DE  E=EDIT":"  >"
260 ACCEPT AT(24,4)VALIDATE("SCNE")SIZE(
1)BEEP:QQ$
270 ON POS("SCNE",QQ$,1)GOSUB 1030,1390,
300,300
280 GOSUB 710
290 GOTO 250
300 GOSUB 880
310 R=1 :: C=1
320 DISPLAY AT(21,3):"WHICH WINDOW? 1-4"
330 CALL KEY(0,X,S)
340 CALL SOUND(1,200,8)
350 IF X>48 AND X<53 THEN 360 ELSE 330
360 X=X-48
370 DISPLAY AT(13,20):"<WINDOW";X
380 GOSUB 1490
390 DISPLAY AT(21,1):"":"KEY: (1-4)=MOVE
 (9,0)=ON/OFF"
400 REM MOVE CURSOR
410 CALL KEY(0,K,S)
420 CALL HCHAR(R+1,C+3,104)
430 IF G(X,R,C)=1 THEN CALL HCHAR(R+1,C+
3,105)ELSE CALL HCHAR(R+1,C+3,101)
440 IF S=0 THEN 410
450 IF K=13 THEN 460 ELSE 480
460 GOSUB 1600
470 GOTO 690
480 IF K>=48 AND K<=57 THEN 490 ELSE 410
490 ON K-47 GOTO 630,520,530,500,510,540
,550,560,570,620
500 C=C-1 :: GOTO 590
510 C=C+1 :: GOTO 590
520 R=R+1 :: GOTO 590
530 R=R-1 :: GOTO 590
540 C=C+1 :: R=R+1 :: GOTO 590
550 C=C+1 :: R=R-1 :: GOTO 590
560 C=C-1 :: R=R+1 :: GOTO 590
570 C=C-1 :: R=R-1
580 REM CURSOR WRAP AROUND
590 IF R=0 OR R=17 THEN 650
```

```
600 IF C=0 OR C=17 THEN 670
610 GOTO 410
620 G(X,R,C)=1 :: GOTO 640
630 G(X,R,C)=0
640 GOTO 410
650 IF R=0 THEN R=16 ELSE R=1
660 GOTO 410
670 IF C=0 THEN C=16 ELSE C=1
680 GOTO 410
690 RETURN
700 REM USED WINDOWS DISPLAY
710 V$=""
720 DISPLAY AT(4,23):"USED"
730 CALL HCHAR(5,23,95,7)
740 CALL HCHAR(7,23,95,7)
750 FOR I=1 TO 4
760 IF A$(I)>RPT$("0",64)THEN V$=V$&STR$
(I)&" "
770 NEXT I
780 DISPLAY AT(6,21):V$
790 RETURN
800 REM LOAD COLORS
810 FOR I=1 TO 16
820 READ C$(I)
830 NEXT I
840 RETURN
850 DATA CLEAR,BLACK,M. GREEN,L. GREEN,D
. BLUE,L. BLUE,D. RED,CYAN,M. RED
860 DATA L. RED,D. YELLOW,L. YELLOW,D. G
REEN,MAGENTA,GRAY,WHITE
870 REM GRAPH SUB
880 CALL CLEAR
890 N$="1234567812345678"
900 FOR R=1 TO 16
910 DISPLAY AT(1+R,1):SEG$(N$,R,1)
920 DISPLAY AT(1+R,18):SEG$(N$,R,1)
930 CALL HCHAR(R+1,4,101,16)
940 NEXT R
950 FOR I=1 TO 2
960 DISPLAY AT(1+D,2):"1234567812345678"
970 D=17
980 NEXT I
990 D=0
1000 GOSUB 710
1010 RETURN
```

```
1020 REM SEE GRAPHICS
1030 CALL CLEAR
1040 PRINT :"MAGNIFY(3 OR 4)":"MOTION(-9
9 TO 99)":"  X AXIS":"  Y AXIS":"SPRITE
COLOR 2-16":"SCREEN COLOR 2-16"
1050 ACCEPT AT(18,19)VALIDATE("34")SIZE(
1)BEEP:M
1060 IF M<3 OR M>4 THEN M=4
1070 ACCEPT AT(20,10)VALIDATE(NUMERIC)SI
ZE(3)BEEP:M1
1080 IF M1<-99 OR M1>99 THEN 1070
1090 ACCEPT AT(21,10)VALIDATE(NUMERIC)SI
ZE(3)BEEP:M2
1100 IF M2<-99 OR M2>99 THEN 1090
1110 ACCEPT AT(22,19)SIZE(2)BEEP:C
1120 IF C<2 OR C>16 THEN C=2
1130 ACCEPT AT(23,19)VALIDATE(NUMERIC)SI
ZE(2)BEEP:SC
1140 IF SC<2 OR SC>16 THEN SC=6
1150 CALL SCREEN(SC)
1160 IF M=4 THEN S=16 ELSE S=0
1170 CALL CHAR(128,A$(1),132,A$(2),136,A
$(3),140,A$(4))
1180 CALL MAGNIFY(M)
1190 CALL SPRITE(#1,128,C,100,66+S)
1200 CALL SPRITE(#2,132,C,116+S,66+S)
1210 CALL SPRITE(#3,136,C,100,82+S+S)
1220 CALL SPRITE(#4,140,C,116+S,82+S+S)
1225 CALL MOTION(#1,M1,M2,#2,M1,M2,#3,M1
,M2,#4,M1,M2)
1230 FOR TM=1 TO 300 :: NEXT TM
1240 CALL CLEAR
1250 DISPLAY AT(22,1):"SPRITE COLOR IS "
;C$(C)
1260 DISPLAY AT(23,1):"SCREEN COLOR IS "
;C$(SC)
1270 REM DISPLAY COLORS DFLT
1280 O=15
1290 IF SC=2 THEN GOSUB 1360
1300 INPUT "MODIFY SPRITE ATTRIBUTES?":Q
$
1310 IF SEG$(Q$,1,1)="Y" THEN 1030
1320 CALL DELSPRITE(ALL)
1330 CALL SCREEN(6)
1340 O=2 :: GOSUB 1360
```

```
1350 RETURN
1360 CALL COLOR(0,0,1,3,0,1,4,0,1,5,0,1,
6,0,1,7,0,1,8,0,1)
1370 RETURN
1380 REM DISPLAY HEX CODES
1390 CALL CLEAR
1400 FOR I=1 TO 4
1410 PRINT :"WINDOW";I;"HEX="
1420 IF A$(I)="" THEN 1430 ELSE 1440
1430 PRINT "    AVAILABLE    " :: GOTO 14
50
1440 PRINT " ";SEG$(A$(I),1,16):" ";SEG$
(A$(I),17,16):" ";SEG$(A$(I),33,16):" ";
SEG$(A$(I),49,16)
1450 NEXT I
1460 PRINT
1470 RETURN
1480 REM CLEAR CHOSEN WINDOW
1490 DISPLAY AT(21,1):"   PLEASE WAIT."
1500 IF A$(X)<="0" THEN 1580
1510 FOR I=1 TO 16
1520 FOR J=1 TO 16
1530 IF QQ$="E" THEN 1540 ELSE 1560
1540 IF G(X,I,J)>0 THEN CALL HCHAR(I+1,J
+3,105)ELSE 1570
1550 GOTO 1570
1560 G(X,I,J)=0
1570 NEXT J :: NEXT I
1580 RETURN
1590 REM WINDOW TO HEX CONV.
1600 HC$="0123456789ABCDEF"
1610 CALL SOUND(333,1110,1)
1620 Z1$,Z2$=""
1630 DISPLAY AT(22,1):" CONVERTING YOUR
HEX CODE"
1640 FOR R=1 TO 16
1650 CD1=G(X,R,1)*8+G(X,R,2)*4+G(X,R,3)*
2+G(X,R,4)+1
1660 CD2=G(X,R,5)*8+G(X,R,6)*4+G(X,R,7)*
2+G(X,R,8)+1
1670 CD3=G(X,R,9)*8+G(X,R,10)*4+G(X,R,11
)*2+G(X,R,12)+1
1680 CD4=G(X,R,13)*8+G(X,R,14)*4+G(X,R,1
5)*2+G(X,R,16)+1
```

```
169Ø Z1$=Z1$&SEG$(HC$,CD1,1)&SEG$(HC$,CD
2,1)
17ØØ Z2$=Z2$&SEG$(HC$,CD3,1)&SEG$(HC$,CD
4,1)
1710 NEXT R
172Ø A$(X)=Z1$&Z2$
173Ø DISPLAY AT(18,2):SEG$(Z1$,1,16):" "
;SEG$(Z1$,17,16):" ";SEG$(Z2$,1,16):" ";
SEG$(Z2$,17,16):""
174Ø RETURN
```



FORMATTED SCREEN

This short listing can be used as a subroutine in your programs. Wherever a program accepts inputs, one can GOSUB to this routine and a colored band will appear to the right of the cursor indicating the acceptable length of the entry. Entries may appear anywhere on the screen and entry length can be controlled. This colored entry format lets the user know the limit of each entry and makes the appearance of the program more attractive.

The routine uses a CALL KEY statement to concanternate key inputs into the variable K$. Just before sending program control to this routine, set the row and column you want the input to appear on into the variables "R" and "C". The length of an entry is assigned to the variable "L". Now, GOSUB to the routine to accept user input. The BACK ARROW and the ENTER keys may be used to edit during all entries. Pressing ENTER accepts an input and returns control back to the main program. The variable K$ now contains the users entry. To save the users input for future use, assign K$ to an appropriate variable that can be accessed later on during the program.

EXAMPLE:

```
100 R=(accept at row)
110 C=(accept at column)
120 L=(max. entry length)
130 GOSUB (routine line#)
140 ANSWER$=K$
```

```
100 REM FORMATTED SCREEN
110 REM CRACKING THE 99/4A
120 CALL COLOR(14,2,7)
130 K$=""
140 GOTO 250
150 CALL KEY(0,K,ST)
160 CALL HCHAR(R,LEN(K$)+6,142)
170 CALL HCHAR(R,LEN(K$)+6,31)
180 IF ST<1 THEN 150
```

```
190 IF K=8 AND LEN(K$)<2 THEN 130
200 IF K<>8 THEN 220
210 K$=SEG$(K$,1,(LEN(K$))-1):: GOTO 250
220 IF K=13 THEN 270
230 K$=K$&CHR$(K)
240 IF LEN(K$)-1=L THEN 210
250 DISPLAY AT(R,C):K$&RPT$(CHR$(143),L-
LEN(K$))
260 GOTO 150
270 RETURN
```

Extended Basic

HORIZONTAL SCROLLING

This is a very simple routine that can be accessed as a subroutine or inserted inbetween program code. It is short and can be useful for displaying messages when space is limited.

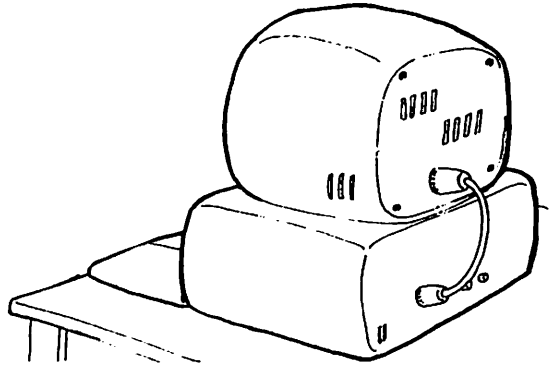
Before sending program control to this routine, assign to the variable ROW the number of the row on which the message is to appear. Next assign the phrase to the variable PHRASE$. The phrase can be as long as can fit when assigning it to the variable. When running the routine, if the scrolling is too fast to read comfortably, insert a FOR NEXT loop or other code between the program lines.

```
100 REM HORIZONTAL SCROLL
110 REM CRACKING THE 99/4A
120 K=1 :: L=28
130 FOR I=28+LEN(PHRASE$)TO 1 STEP -1
140 IF L>1 THEN J=J+1 ELSE K=K+1
150 DISPLAY AT(ROW,L):SEG$(PHRASE$,K,J)
160 L=L-1
170 NEXT I
```

VERTICAL PRINT

This routine allows you to print messages on your screen vertically. Set the variable WORD$ with a phrase. Be careful not to let it exceed 24 rows and spaces. The message starts on row one. The column for display can be set in the variable COL. If you want to start on another row, change the variable ROW= in line 12Ø. Be sure not to let the message exceed row 24.

```
1ØØ REM PRINT VERTICALLY
11Ø REM CRACKING THE 99/4A
12Ø FOR ROW=1 TO LEN(WORD$)
13Ø CALL VCHAR(ROW,COL,ASC(SEG$(WORD$,ROW,1
)))
14Ø NEXT ROW
```



FILE I/O ASSEMBLY ROUTINES

MANIPULATING DISK FILES IN ASSEMBLY LANGUAGE

Casual users of the 99/4A may never need to use disk files in assembly language. This is because both Ti Basic and Ti Extended Basic contain a wealth of support calls for file handling. Many types of algorithms may be implemeted using these two languages.

Care must be used in choosing which algorithms to use due to the relatively slow process of using a Basic Interpreter on the 99/4A. Sequential searches should be avoided with long files! Many program applications can be coded with an acceptable execution time.

But what about the person who demands a little preformance out of his $100 machine? What about the person who is programming games and needs to access records of bit map mode pattern data, sprite pattern data, speech data, and sound data? Basic is much to slow to serve in these capacities.

Programming in assembly provides the key. A typical instruction on the 99/4A takes only a few micro seconds to execute. Thus many instructions can be executed very quickly, relative to that of the basic interpreter. But unlike programming in Basic, the assembly language programmer is faced with the task of providing the system services that are provided transparently by the Basic interpreter.

For file manipulation this means the programmer must be aware of what a PAB is, and how to sucessfully call and use the Rom & Grom routines at his disposal. Certainly the programmer must be experienced in 9900 assembly language programming!

The following program was developed as one of the first modules in an assembly langauge programming arsenal. It is a very simple rendition of file service calls for use by any application. The routines include a general purpose file I/O call, which is used to open, read, write, and close a file. A general purpose routine is included for determining if any errors were encountered during DSR execution. Three routines were designed to provide Ram and Vdp Ram buffer management. Finally, a routine is included to read the file status of a device.

An application uses the routines by including a REF for each routine that requires access, and also defines a FCB for each file needed. The Editor assembler is then used to load the file I/O object module into memory after your application program. This method is used to speed up development of the application program. This is achieved via seperate compilation of the file i/o routines, thus avoiding recompiling the file I/O routines each time the application program is re-compiled. This also saves printing time, as few of us can afford a truely fast printer(or the reams of paper)!

```
       TITL 'F I L E   I / O'
*********************************************
*                                           *
* MODULE    : FILEIO                        *
* PURPOSE   : Provide file services to      *
*             the 994/a programmer          *
* AUTHOR    : Joel Rodriguez                *
* SITE      : Austin, Texas                 *
* REGISTERS: RØ-R2 : Scratch                *
*            R8    : Error return address   *
*            R9    : Fcb Base Address       *
*            R1Ø   : Ram Base Address       *
* HISTORY   : Original 12/83                *
*                                           *
* ALL RIGHTS RESERVED BY AUTHOR 1983        *
*                                           *
*********************************************
       DEF  IOFLE,FLECHK,G$VDP,P$VDP,CLRRAM,G$STAT
       REF  VMBW,DSRLNK,VSBR,VMBR
*
*      General Equates
*
REG1   EQU  2              OFFSET OF R1 FROM WP
RETADR EQU  8              R8 = ERROR RETURN ADDR
FCB    EQU  9              R9 = FCB ADDRESS
RAMADR EQU  1Ø             R1Ø= RAM ADDRESS
DSRPTR EQU  >8356          DSR I/O POINTER
*                                           *
*===========================================*
* IOFLE : Routine to write the file Pab to  *
*         Vdp Ram, and issue call to the    *
*         Dsr.                              *
*===========================================*
IOFLE  EQU  $              ENTRY POINT
       MOV  *R11+,FCB
*
       MOV  @VDPADR(FCB),RØ   Write the pab to vdp
       MOV  FCB,R1
       AI   R1,PAB
       MOV  @PABLEN(FCB),R2
       BLWP @VMBW
*
       MOV  @VDPADR(FCB),RØ   Format address for dsr
       AI   RØ,DESCLN
       MOV  RØ,@DSRPTR
```

```
       BLWP @DSRLNK           Do file I/O
       DATA 8
       RT                     Return
*
*=============================================*
* FLECHK: Routine to read the file Vdp Pab  *
*         and report any errors.              *
*=============================================*
FLECHK EQU  $                 ENTRY POINT
       MOV  *R11+,RETADR      Store error address
       AI   RØ,STATUS         Add status offset
       STWP R1                Get workspace address
       AI   R1,REG1           Add offset to R1
       BLWP @VSBR             Read status into R1
       CLR  RØ
       MOVB R1,RØ             Copy status for testing
       MOV  RØ,R2
       SLA  RØ,3            ? Any of 1st 3 bits set
       JOC  FLEMSG            Yes, go report error
       RT
*
*=============================================*
* FLEMSG: Routine to output error messages. *
*=============================================*
FLEMSG SRL  R2,13             Justify
       SLA  R2,1                make index
       MOV  @MSGTBL(R2),R1    Load message address
*
       LI   RØ,>1Ø2           TEMP SOLUTION
       LI   R2,28               TEMP SOLUTION
       BLWP @VMBW                 TEMP SOLUTION
       MOV  RETADR,R11
       RT
*
ERR1   TEXT 'Error, file write protected '
ERR2   TEXT 'Error, in file open         '
ERR3   TEXT 'Error, illegal file op      '
ERR4   TEXT 'Error, table/buffer overflow'
ERR5   TEXT '  **  End of file  **       '
ERR6   TEXT 'Error, device error         '
ERR7   TEXT 'Error, file type            '
OHOH   TEXT 'Error, programming          '
*
```

```
MSGTBL DATA MSG$E-$
       DATA ERR1
       DATA ERR2
       DATA ERR3
       DATA ERR4
       DATA ERR5
       DATA ERR6
       DATA ERR7
MSG$E  DATA OHOH
*
*=============================================================*
* G$VDP  : XFER Vdp ram buffer to ram buffer                  *
* TO CALL: Load RØ with fcb address                           *
*          Do a BL   @G$VDP                                   *
*               DATA BUFFER                                   *
*          where BUFFER is the ram buffer address             *
* RETURNS: XFER count in RØ                                   *
* REGS   : RØ - Input word count                              *
*=============================================================*
G$VDP  EQU  $                 ENTRY POINT
       MOV  *R11+,RAMADR      Save ram buffer address
       MOV  RØ,FCB            Save file fcb address
       AI   RØ,VDPADR
       MOV  *RØ,RØ            Get vdp address
       AI   RØ,COUNT            of record count
       CLR  R1
       BLWP @VSBR             Get record count
       SWPB R1
       MOV  R1,@XFCT          Save XFER count
       MOV  R1,R2
       MOV  RAMADR,R1
       AI   FCB,PAB+BUFADD
       MOV  *FCB,RØ
       BLWP @VMBR             Get the record
       MOV  @XFCT,RØ          Return the XFER count
       RT
*
XFCT   DATA Ø                 Temp xfer count storage
*=============================================================*
* P$VDP  : XFER ram buffer to VDP ram buffer                  *
* TO CALL: Load RØ with fcb address                           *
*          Load R1 with XFER count                            *
*          Do a BL   @P$VDP                                   *
*               DATA BUFFER                                   *
*          where BUFFER is the ram buffer address             *
```

```
* REGS    : R0 - Input word count                          *
*           R1 - Input xfer count                          *
*=========================================================*
P$VDP   EQU  $                  ENTRY POINT
        MOV  *R11+,RAMADR       Fetch Ram address
        MOV  R0,FCB             Store Fcb address
        MOV  R1,R2
        MOV  RAMADR,R1
        AI   FCB,PAB+BUFADD
        MOV  *FCB,R0
        BLWP @VMBW              Write Vdp Ram
        RT                      Return
*
*=========================================================*
* G$STAT : Obtain the file status byte of the Pab.       *
* TO CALL: Load R0 with Vdp Pab Address                  *
*          Do a BL   @G$STAT                             *
*          On return R0 contains the status byte in      *
*          the Msb byte.                                 *
* REGS : R0 - Vdp Pab address/File status                *
*=========================================================*
G$STAT  EQU  $              ENTRY POINT
        AI   R0,FLESTS      Add file status offset
        STWP R1
        AI   R1,REG1        Add R1 offset from Wp
        BLWP @VSBR          Fetch file status byte
        CLR  R0
        MOVB R1,R0
        RT                  Return in MSB of R0
*
*=========================================================*
* CLRRAM : Clear ram buffer                              *
* TO CALL: Load R0 with word count                       *
*          Do a BL   @CLRRAM                             *
*               DATA BUFFER                              *
*          where BUFFER is the ram buffer address        *
*                                                        *
* REGS : R0 - Input word count                           *
*=========================================================*
CLRRAM  EQU  $                  ENTRY POINT
        MOV  *R11+,R1           Get ram address data
        MOV  R1,R2
        A    R0,R2              Format ram end address
        CLR  R0
CLRLP   MOV  R0,*R1+            Clear ram word
```

```
        C     R1,R2           ? End address reached
        JNE   CLRLP             No, go do it again
*
        RT
*
*****************************************************
*
*
*       FCB DORG
*
        DORG Ø
PABLEN  DATA Ø                  Length of Pab
VDPADR  DATA Ø                  Pab vdp ram address
PAB     DATA Ø,Ø,Ø,Ø            PAB
        BYTE Ø
N$LEN   BYTE Ø                  Length of file name
F$NAM   BSS  15                 File name text
RSERV   BYTE Ø
        RORG
*
*****************************************************
*
*
*       PAB BLOCK TO DSR's
*
        DORG Ø
OPCODE  BYTE Ø              I/O OPCODE
STATUS  BYTE Ø              DSR STATUS BYTE
BUFADD  DATA Ø              DATA BUFFER ADDRESS
LRL     BYTE Ø              LOGICAL RECORD LENGTH
COUNT   BYTE Ø              XFER COUNT
RECNUM  DATA Ø              REL REC NUMBER
FLESTS  BYTE Ø              FILE STATUS
DESCLN  BYTE Ø              NAME DESCRIPTOR LENGTH
        RORG
*****************************************************
```

DESCRIPTION OF THE FILE I/O ENTRY POINTS :

IOFLE— This routine writes the pab to Vdp Ram, and issues a call to the DSR. No verification of the pab is performed.

FLECHK— This routine reads the file pab status byte from vdp ram. If an error is detected error action is taken including returning control of the program to an error address.

G$VDP— Transfers a buffer of Vdp Ram to Ram memory.

P$VDP— Transfers a buffer of Ram memory to Vdp Ram.

G$STAT— Reads the file status byte of the file pab in Vdp Ram.

The copy files FCB/SRC and PAB/SRC contain DORG's which define the offsets of the various elements with the data structures. The copy files are handy ways of maintaining some unity of equates among various modules.

The Fcb copy file contains :

```
*       FCB DORG
*
        DORG Ø
PABLEN  DATA Ø                  Length of Pab
VDPADR  DATA Ø                  Pab vdp ram address
PAB     DATA Ø,Ø,Ø,Ø            PAB
        BYTE Ø
N$LEN   BYTE Ø                  Length of file name
F$NAM   BSS  15                 File name text
RSERV   BYTE Ø
        RORG
*
*************************************************
```

The Pab copy file contains :

```
*
*       PAB BLOCK TO DSR's
*
        DORG Ø
OPCODE  BYTE Ø              I/O OPCODE
```

```
STATUS BYTE Ø          DSR STATUS BYTE
BUFADD DATA Ø          DATA BUFFER ADDRESS
LRL    BYTE Ø          LOGICAL RECORD LENGTH
COUNT  BYTE Ø          XFER COUNT
RECNUM DATA Ø          REL REC NUMBER
FLESTS BYTE Ø          FILE STATUS
DESCLN BYTE Ø          NAME DESCRIPTOR LENGTH
       RORG
```

Next some background! A PAB (Peripheral Access Block) is a data structure which is used to pass information between one's program logic and the Rom Dsr routines of the 99/4A. The Pab is 5 words long, plus 1 byte for each character in the file pathname. Thus the Pab for the file named "DSK1.TEST" is 5 words plus 9 bytes, or 19 bytes. The Pab must be located in Vdp Ram before a call to the Dsr may be made. It is this programming consideration that makes the use of a file i/o module both a code and time saver!

Vdp Ram is that block of memory that is managed by the TMS9918A Video Display Processor Chip. This chip was used to cut the costs associated with the internal hardware memory management architecture. It is the chip that provides all the reasons I bought the 99/4A. GRAPHICS. But, it is also the cause for some not so standard programming techniques. That is, the programmer must be aware that he must treat Vdp memory differently than that normal for expansion ram.

The requirement for the FCB(File Control Block) is thus born. The FCB resides in readily accessable ram. It contains the Pab, and also includes two additonal words; the total length of the FCB and the Vdp address for the Pab. The Fcb is used to pass data to the file I/O routines and finally to Vdp Ram. The FCB is more important in concept than in implementation details. Readers are encouraged to develop their own Fcb's if needed.

The major factor in constructing the file I/O routines is the decision to maintain the various file Pab's in Ram, then copying the Pab to Vdp Ram each time a file I/O request is made. This allows accidental destruction of Vdp Ram to occur while assuring that Vdp ram is correct during file i/o. This benefit should be obvious to those having attempted assembly langauge programming on the 99/4A!

If you like to optimize, or must optimize the execution speed of the file I/O routines you may choose to maintain your file Pab's in Vdp Ram. The support routines would then include provisions for modifying single bytes of the Pab on the fly. This approach was not initially coded due to the slightly longer code generation cycle, and the uncertain cloud that is telling me I may not need any extra speed!

And finally I have coded an example using the file I/O routines. This example was intended to help provide some insight into the structure of the Basic Merge File; in hopes of finding a key to allow program development without the Basic Editor. The idea is to allow generation of basic text files that could be converted to Basic program images. The Basic merge file just happens to be void of tagged object code for the interpreter. The elimination of that hurdle was encouraging.

But, the example clearly demonstrates the use of the file I/O routines. Be forwarned. The logic shown here does not provide for formatting of records longer than 80 characters, and are lost! But you didn't want to look at all of the merge file anyway, did you? Also no attempt was made to format I/O to the screen, that is best left for your tastes and needs.

```
*****************************************
*                                       *
* MODULE : INSMRG                       *
* PURPOSE: Inspect a MERGED BASIC       *
*          PROGRAM FILE                 *
* AUTHOR : Joel Rodriguez               *
* SITE   : Austin, Texas                *
* HISTORY: Original 12/83               *
*                                       *
* ALL RIGHTS RESERVED BY AUTHOR 1983    *
*                                       *
*****************************************
       TITL 'INSPECT MERGE FILE'
*
*      DSRLNK Command Equates
*
OPEN   EQU  >0000        OPEN FILE
CLOSE  EQU  >0100        CLOSE FILE
READ   EQU  >0200        READ RECORD
WRITE  EQU  >0300        WRITE RECORD
RESTOR EQU  >0400        FILE RESTORE
LOAD   EQU  >0500        LOAD IMAGE FILE
SAVE   EQU  >0600        SAVE IMAGE FILE
DELETE EQU  >0700        DELETE FILE
SCRTCH EQU  >0800        SCRATCH REL RECORD
RDSTAT EQU  >0900        READ FILE STATUS
*
*
*
TXTFLE EQU  >0012        OPEN TEXT FILE
MERGE  EQU  >0014        OPEN MERGE FILE
```

```
OPNPGM EQU  >0004          OPEN FILEIO OBJECT
*
*       VDP Ram Equates
*
TXTPAB EQU  >F80           IMAGE FILE PAB
MRGPAB EQU  >FA0           TEXT FILE PAB
MSGPAB EQU  >FC0           MESSAGE FILE PAB
TXTBUF EQU  >1000          TEXT RECORD BUFFER
MRGBUF EQU  >1100          MERGE RECORD BUFFER
MSGBUF EQU  >1200          MESSAGE RECORD BUFFER
*
*       Program equates and data area
*
USERWP EQU  >20BA          PROGRAM WORKSPACE
BUF$TX BSS  >100           TEXT FILE BUFFER
BUF$IM BSS  >100           MERGE FILE IMAGE BUFFER
BUF$MG BSS  >100           MESSAGE FILE BUFFER
R11SAV DATA 0              RETURN ADDRESS STORAGE
RDCT   DATA 0              INPUT RECORD COUNT
WRTCT  DATA 0              OUTPUT RECORD COUNT
LSTREC DATA 0              LAST RECORD READ FLAG
*
*
*       File Control Blocks (FCB)
*
       EVEN                    MERGE FILE CONTROL BLOCK(FCB)
MRGFCB EQU  $
       DATA TAILM-PABMRG
       DATA MRGPAB
PABMRG DATA MERGE          MERGE FILE DSR PAB
       DATA MRGBUF
       DATA MRGLRL,0
       BYTE 0
       BYTE TAILM-MRGNAM
MRGNAM TEXT 'DSK1.MERGET/IN  '  NAME OF IMAGE FILE
TAILM  EQU  $-2
*
*
*
       EVEN                    TEXT FILE CONTROL BLOCK(FCB)
TXTFCB EQU  $
       DATA TAILT-PABTXT
       DATA TXTPAB
PABTXT DATA TXTFLE          TEXT FILE DSR PAB
       DATA TXTBUF
```

```
        DATA TXTLRL,0
        BYTE 0
        BYTE TAILT-TXTNAM          TEXT FILE PAB BLOCK
TXTNAM  TEXT 'DSK1.MERGET/OUT '    NAME OF TEXT FILE
TAILT   EQU  $-1
*
*       Fcb equates
*
IMGLRL  EQU  >0000          MERGE FILE LRL(DSR RETURNS)
TXTLRL  EQU  >5000          TEXT FILE LRL
MRGLRL  EQU  163            MERGE FILE LRL IS 163
*
*       Register Equates
*
BASADR  EQU  3              BASE ADDRESS REGISTER
TXTPTR  EQU  4              TEXT BUFFER POINTER REGISTER
FCBPTR  EQU  5              FILE CONTROL BLOCK POINTER
BUFPTR  EQU  6              BUFFER POINTER REGISTER
RECNM   EQU  7              RECORD NUMBER REGISTER
REG1    EQU  2              R1 ADDRESS OFFSET FROM WP
*
GPLSTS  EQU  >837C          GPL STATUS BYTE
HEADER  TEXT 'REC#'         OUTPUT TEXT STRING
*

*       Program defs/refs
*
        DEF  INSMRG
        REF  IOFLE,FLECHK,P$VDP,G$VDP,CLRRAM,G$STAT
        EVEN
*******************************************************
*       Program entry
*******************************************************
INSMRG  MOV  R11,@R11SAV
        LWPI USERWP
*
*       Open a merged program file
*
        BL   @IOFLE
        DATA MRGFCB
*
        LI   R0,MRGPAB          Check for errors
        BL   @FLECHK
        DATA F$EXT              Error recovery address
*
```

```
*       Open text output file
*
        BL    @IOFLE
        DATA  TXTFCB
*
        LI    RØ,TXTPAB         Check for errors
        BL    @FLECHK
        DATA  F$EXT             Error recovery address
*
*       Clear Ram record buffers
*
        LI    RØ,>1ØØ
        BL    @CLRRAM           Clear text buffer
        DATA  BUF$TX
        LI    RØ,>1ØØ
        BL    @CLRRAM           Clear Image buffer
        DATA  BUF$IM
        LI    RØ,>1ØØ
        BL    @CLRRAM           Clear Msg buffer
        DATA  BUF$MG
        PAGE
*
*       Read a merge file record into Ram
*
        CLR   RECNM             R7 = WRITE RECORD NUMBER
RDREC   LI    TXTPTR,BUF$TX     R4 = @BUF$TX
        LI    FCBPTR,MRGFCB     R5 = @MERGE FILE FCB
        LI    BUFPTR,BUF$IM     R6 = @BUF$IM
*
        LI    BASADR,READ
        MOVB  BASADR,@PAB(FCBPTR)  Update fcb to read
        BL    @IOFLE
        DATA  MRGFCB
*
        LI    RØ,MRGPAB         Check for errors
        BL    @FLECHK
        DATA  F$MRG
*
        LI    RØ,MRGFCB
        BL    @G$VDP            Xfer record to ram
        DATA  BUF$IM            Count returned in RØ
        MOV   RØ,@RDCT          Save count
*
        LI    BASADR,RDSTAT
        MOVB  BASADR,@PAB(FCBPTR)  Change Pab to read status
```

```
        BL    @IOFLE
        DATA MRGFCB              Read the file status
*
        LI    RØ,MRGPAB
        BL    @FLECHK            Check for dsr errors
        DATA F$MRG
*
        CLR   @LSTREC
        LI    RØ,MRGPAB
        BL    @G$STAT            Go get file status
        ANDI RØ,>1ØØ           ? Last record read
        JEQ   WRTREC             No, go continue
        SETO @LSTREC             Set last record flag
        JMP   NXTCHR             Go process last two bytes
        PAGE
*
*       Format text ram buffer
*
WRTREC CLR   @WRTCT              Clear write byte count
        LI    BASADR,HEADER
        MOV   *BASADR+,*TXTPTR+ Output 'REC#'
        MOV   *BASADR+,*TXTPTR+
        INCT @WRTCT              Bump write count
        INCT @WRTCT
*
        MOV   @RDCT,RØ           Get input count
        CI    RØ,2               ? More than 2 bytes
        JLE   NXTCHR             No, then skip rec process
        MOV   *BUFPTR+,RØ        Process record number
        BL    @HEXASC            Convert two bytes
        DATA 2
        DECT @RDCT
NXTCHR
        CLR   RØ
        MOVB *BUFPTR+,RØ         Get a byte
        CI    RØ,>2ØØØ           ? Ascii lower bound
        JHE   HITST              Yes, check upper bound
        JMP   CONVRT
HITST
        CI    RØ,>6EØØ           ? Ascii upper bound
        JLE   DATAOK             Yes, go write the byte
CONVRT
        BL    @HEXASC            Convert byte to Ascii Word
        DATA 1
        JMP   ENDBUF
```

```
DATAOK
        MOVB R0,*TXTPTR+        Write the Ascii character
        INC  @WRTCT
ENDBUF
        DEC  @RDCT             ? End of image byte stream
        JNE  NXTCHR              No, continue processing
*
*       Set up Vdp ram, TXTFCB, and write the record
*
        LI   FCBPTR,TXTFCB
        LI   BASADR,WRITE
        MOVB BASADR,@PAB(FCBPTR)  Set opcode to write
*
        MOV  FCBPTR,R0
        AI   R0,PAB+COUNT
        SWPB @WRTCT               Modify word to byte data
        MOVB @WRTCT,*R0           Set write count in Pab
        LI   R0,TXTFCB
        SWPB @WRTCT               Restore count as data word
        MOV  @WRTCT,R1
        BL   @P$VDP               Write the ram buffer
        DATA BUF$TX
*
        BL   @IOFLE               Write the text record
        DATA TXTFCB
*
        LI   R0,TXTPAB            Check for errors
        BL   @FLECHK
        DATA F$EXT
*
        INC  RECNM
        ABS  @LSTREC           ? Last input record
        JEQ  RDREC               No, process again
*
*       Close output file and exit
*
F$EOF
F$MRG
        LI   FCBPTR,TXTFCB
        LI   BASADR,CLOSE
        MOVB BASADR,@PAB(FCBPTR)
        BL   @IOFLE                Close the test file
        DATA TXTFCB
F$EXT
        CLR  R0
```

```
       MOVB R0,@GPLSTS          Clear Gpl status
       MOV  @R11SAV,R11         Restore return address
       RT                       RETURN TO CALLER
*
       TITL 'SUBROUTINES'
*****************************************************
*      SUBROUTINES                                  *
*****************************************************
* HEXASC : Formats decimal values to printable
*          Ascii, and writes Ascii stream to that
*          pointed to by TXTPTR.
* TO CALL: Load R0 with hex data word, byte data
*          data in Msb.
*          BL    @HEXASC
*          DATA  X  where x is 1 or 2 byte conversion
*
TOKEN  EQU  8
NIBBS  DATA 1                   Number of nibbles to convert
HEXSAV DATA 0                   Return address storage
HEXCHR TEXT ' >'
*
HEXASC EQU  $
       MOV  *R11+,@NIBBS
       MOV  R11,@HEXSAV         Save return address
       MOV  R0,TOKEN            Save token byte
       LI   BASADR,HEXCHR
       MOVB *BASADR+,*TXTPTR+
       MOVB *BASADR,*TXTPTR+    Output ' >'
       INCT @WRTCT              Bump write count
HEX$LP
       MOV  TOKEN,R0
       ANDI R0,>F000            Process msb nibble
       SRL  R0,4
       BL   @ADJUST
       MOVB R0,*TXTPTR+
       MOV  TOKEN,R0            Process lsb nibble
       BL   @ADJUST
       MOVB R0,*TXTPTR+
       INCT @WRTCT
       MOV  @NIBBS,R1
       CI   R1,1              ? Last nibble
       JEQ  HEXEND              Yes, then exit
       DEC  @NIBBS
       SWPB TOKEN
       JMP  HEX$LP
```

```
HEXEND MOV  @HEXSAV,R11      Restore return address
       RT
*
*********************************************************
* ADJUST : Adjust hex nibble to an Ascii byte.
*          Nibble is in bits 4-7 of RØ.  The Ascii byte
*          is returned as the Msb of RØ.
*
BAS    EQU  >Ø9ØØ          Compare value
BAS9   EQU  >3ØØØ          Increment to Base Ascii 'Ø'
BASA   EQU  >37ØØ          Increment to Base Ascii 'A'
NMASK  EQU  >ØFØØ          Nibble Mask
ADJUST EQU  $
       ANDI RØ,NMASK       Mask unwanted bits
       CI   RØ,BAS      ? Nibble > 9
       JGT  TENADJ         Yes, adjust to Ascii 'A'
       AI   RØ,BAS9        Adjust based at >3Ø, Ascii '9'
       JMP  ADJEND           and return
TENADJ AI   RØ,BASA        Adjust based at >41, Ascii 'A'
ADJEND RT                  Return
*
*********************************************************
*
*
*      PAB BLOCK TO DSR's
*
       DORG Ø
OPCODE BYTE Ø              I/O OPCODE
STATUS BYTE Ø              DSR STATUS BYTE
BUFADD DATA Ø              DATA BUFFER ADDRESS
LRL    BYTE Ø              LOGICAL RECORD LENGTH
COUNT  BYTE Ø              XFER COUNT
RECNUM DATA Ø              REL REC NUMBER
FLESTS BYTE Ø              FILE STATUS
DESCLN BYTE Ø              NAME DESCRIPTOR LENGTH
       RORG

*
*      FCB DORG
*
       DORG Ø
PABLEN DATA Ø                  Length of Pab
VDPADR DATA Ø                  Pab vdp ram address
PAB    DATA Ø,Ø,Ø,Ø            PAB
       BYTE Ø
```

```
N$LEN   BYTE Ø                  Length of file name
F$NAM   BSS  15                 File name text
RSERV   BYTE Ø
        RORG
*
************************************************
```

The use of IOFLE is straight forward. It is the job of the application to set up and maintain the various Fcb's. The call to IOFLE is made with the address of the appropriate Fcb passed as a data item following the call.

```
BL    @IOFLE
DATA FCBADR
```

All good programmers check their external calls for errors after execution. The FLECHK routine is called by loading RØ with the Vdp Ram address of the Pab, calling the routine followed by an error return address data word.

```
LI    RØ,RAMPAB
BL    @FLECHK
DATA ERRRET
```

The details of adding screen i/o to the file i/o routines is left for the industrious.

To issue a read or a write to the file after you have opened it, just modify the RAM pab to reflect the needed operation and call IOFLE again.

Use the CLRRAM routines to clear out any ram buffers you are using. After all, all good programmers intialize their data areas! To use CLRRAM just load RØ with the number of WORDS you wish to clear, and call the routine follwed by the data address of the ram buffer.

```
LI    RØ,>1ØØ
BL    @CLRRAM
DATA RAMADR
```

To get a buffer of data from Vdp Ram to ram memory use the G$VDP routine. Load RØ with the Fcb address, and call the routine with the ram address following the call.

```
LI    RØ,FCBADR
BL    @G$VDP
DATA  RAMADR
```

Conversly, to transfer a buffer from ram memory to Vdp Ram use the P$VDP routine. Load RØ with the Fcb address, load R1 with the buffer byte count, and call P$VDP with the address of the ram data buffer following the call.

```
LI    RØ,FCBADR
LI    R1,LENGTH
BL    @P$VDP
DATA  RAMADR
```

The status of any file may be obtained by calling G$STAT. This will tell you if the file is a program file, or a text file. It will also tell you if the disk is full, or the file is currently at it's last written record. To call the routine, load RØ with the Vdp Ram address of the file, and call the routine. The file status will be returned in RØ.

```
LI    RØ,VDPADR
BL    @G$STAT
```

...END OF TRANSMISSION...

PROGRAM ORDER FORM

Most of the programs in this book are available on diskette. All orders may be sent to:

MIDNIGHT EXPRESS PUBLISHERS
BOX 26941
AUSTIN, TEXAS 78755

The programs completely fill the diskette and are listed below. Books are $12.95, and diskettes are $6.95. Add one dollar shipping per book.

ADDRESS-BASIC TUTORIAL
CHECK MANAGEMENT/ANALYSIS
CHRISTMAS BILL BOARD
CHECKERS FOR 16K
GRAPHICS GENERATOR
HANGMAN FOR TWO
OTHELLO
PHONETIC SPEECH EDITOR
SEEK AND FIND PUZZLE GENERATOR
SPEECH CONTROL PROGRAM
SUPER CATALOGGER
TANK ABOUT MATH

C O R R E C T I O N S

There are a few necessary changes.
The last example on page 31, and the first example on page 32 reads.

```
IF -(((A=2 + B=4)<0)*((A=2 * B=4)=0))<> 0 THEN ...
```

Should read:

```
IF -(((A=2)+(B=4)<0)*((A=2)*(B=4)=0))<>0 THEN ...
```

Excuse me for allowing you to struggle through the books complex sample of exclusive OR. Here is a simple example.

```
IF (A=2)-(B=4)=0 THEN ...
```

Delete four lines on page 34 that read;

I will finish this section with something simple. These two statements are equivalent.

```
EXT.  IF B>10 AND B<20 THEN PRINT "SUCCESS"
REG.  IF 10<B<20 THEN PRINT "SUCCESS"
```

The Basic tutorial program 'ADDRESS' has been modified on the diskette to run using disk also. Line numbers are changed as follows.

Additional line numbers;
121 122 165 400-404 835 876 1171 1401-1409
1511-1519 1665 1701 1751 1761 2161 2021-2063

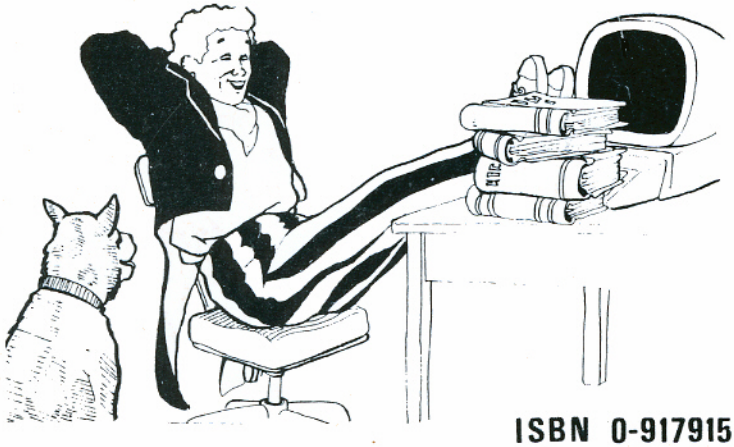
Changed line numbers;
190 430 770 772 780 790 800 820 860 930 1170
1180 1200-1230 1700 1702 1710 1730-1750 1760
2020 2160 2190

Moved line numbers;
405 772 875 1702

$12.95 USA

INSIDE THE BOOK!

* A speech chapter that tells it like it is.
* A game of checkers for 16K.
* A tutorial on logical operators.
* Seek and find puzzle generator—an infinite number of puzzles and you create your own word lists.
* A great grafix editor that can display your designs and their hex code equivalents for an area up to 32x32 pixels (1024 pixels!).
* Take a look inside!



ISBN 0-917915-00-3